Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE – 0502 Proyecto Eléctrico

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA MEDIDOR DE VARIABLES ELÉCTRICAS

Por:
Manuel Guillén Girón

Ciudad Universitaria Rodrigo Facio
Junio de 2008

Diseño e implementación de un sistema medidor de variables eléctricas

Por:
Manuel Guillén Girón

Sometido a la Escuela de Ingeniería Eléctrica
de la Facultad de Ingeniería
de la Universidad de Costa Rica
como requisito parcial para optar por el grado de:

BACHILLER EN INGENIERÍA ELÉCTRICA

Aprobado por el Tribunal:

_____
Ing. Peter Bernal Zeledón Méndez
Profesor Guía

_____
Ing. Gerardo Castro Jiménez
Profesor lector

_____
Ing. Andrew Sheehy Protti
Profesor lector

DEDICATORIA

A mis padres, que a pesar de todas las dificultades de la vida siempre están dispuestos a darlo todo por sus hijos, a mis hermanos por el apoyo dado y sobre todo a Dios.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS.....vi

ÍNDICE DE TABLAS.....vii

NOMENCLATURA.....viii

RESUMEN.....ix

1.CAPÍTULO 1: Introducción.....1

1.1Objetivos.....2
Objetivo general.....2
1.1.2 Objetivos específicos.....2
1.2 Metodología.....2

CAPÍTULO 2: Desarrollo teórico.....5

2.1 Sistemas de instrumentación digital.....5
2.2 Convertidores analógico-digital.....7
2.2.1 El CAD.....7
2.3 Multímetro digital.....10
2.3.1 Medición de voltaje. 10
2.3.2 Medición de corriente.....12
2.3.3 Medición de resistencias.....14
2.3.4 Medidas de capacitancia. 14
2.3.5 Medidas de frecuencia.....15

CAPÍTULO 3: Diseño del circuito de adquisición.....16

3.1 Diseño del circuito de adquisición.....16
3.2 Diseño del voltímetro. 18
3.3 Diseño del amperímetro. 19
3.4 Diseño del ohmímetro. 21
3.5 Diseño del frecuencímetro.....22
3.6 Diseño del capacitímetro. 25
3.7 Diseño de la interfaz del circuito analógico con el módulo USB.....26

CAPÍTULO 4: El estándar USB.....28

4.1 Introducción. 28
4.2 Arquitectura y terminología. 30
4.3 Especificaciones eléctricas y de energía. 31

4.4 Ancho de banda. 31
4.5 Tipos de transferencia y "endpoints”. 32
4.6 El proceso de enumeración e inicialización....34
CAPÍTULO 5: El CY7C68013....37
5.1 Descripción del CY7C68013....37
5.2 El controlador 8051 embebido....41
5.3 El SIE....42
5.4 Tarjeta de desarrollo PmodUSB2....43
CAPÍTULO 6: Implementación y análisis de resultados....45
6.1 Firmware....45
6.1.1 El CY7C68013A....45
6.1.2 Firmware de control del contador....48
6.1.3 Firmware de control del CAD....50
6.1.4 Firmware de depurado....52
6.2 Tarjeta de adquisición de datos. 52
6.3 Etapa de conversión analógica digital....54
6.4 Ejecución y problemas encontrados....56
CAPÍTULO 7: Conclusiones y recomendaciones....61
7.1 Conclusiones....61
7.2 Recomendaciones....63
BIBLIOGRAFÍA....65
APÉNDICES....68
Apéndice A: Firmware....68

ÍNDICE DE FIGURAS

Figura 2.2.1 Error de cuantización de un CAD ideal.....8

Figura 2.2.2 Códigos perdidos.....9

Figura 2.3.1. Divisor de voltaje.....11

Figura 3.1.1 Diagrama del INA2331.....17

Figura 3.2.1 Circuito acondicionador para el voltímetro.....19

Figura 3.4.1 Fuente de corriente constante.....22

Figura 3.5.1 Acondicionador de señal para el circuito contador.....24

Figura 3.5.2 Circuito contador.....24

Figura 3.6.1 Circuito para la medida de capacitancias 25

Figura 5.1.1 Diagrama simplificado del CY7C68013A en su versión de 56 pines.....37

Figura 5.1.2 Diagrama simplificado del CY7C68013A en su versión de 56 pines.....40

Figura 5.4.1 Fotografía y diagrama de conexión del módulo USB 2.0 de Digilent.....43

Figura 5.4.2 "Pin-out" del CY7C68013A en su versión de 56 pines.....44

Figura 6.1.1 Diagrama de temporización del CAD en modo READ.....51

Figura 6.2.1: Salida del multivibrador para a) Cx = 0,1 µF b) Cx= 0,01 µF.....53

Figura 6.2.2: Salida del circuito acondicionador de señal para a) una onda triangular de 125 kHz b) una onda sinusoidal de 1.3 MHz.....54

Figura 6.3.1: Esquemático del filtro pasobajo implementado.....55

Figura 6.3.2: Gráfico de magnitud vs frecuencia para el filtro paso bajo implementado.....55

ÍNDICE DE TABLAS

Tabla 3.3.1 Rangos de corriente del amperímetro 20
Tabla 3.4.1 Valores de resistencias para las escalas del ohmímetro.....22

NOMENCLATURA

a.C.	Amplificador de instrumentación
CAD	Conversor analógico digital
DAT	Distorsión armónica total
FET	Field effect transistor (Transistor de efecto de campo)
FIFO	First in first out (Primero en entrar, primero en salir)
GPIF	General programable interfase (Interfaz programable general)
IEM	Interferencia electromagnética
mps	Muestras por segundo
PCB	Placa de circuito base
ppm	Partes por millón
RMS	Root medium square (Raíz media cuadrática)
RSR	Relación señal ruido
USB	Bus serial universal

RESUMEN

El presente trabajo tuvo como principal objetivo el diseñar una tarjeta que permitiese tomar medidas de variables eléctricas y poder enviarlas a través del puerto USB.

Para lograr este objetivo el proyecto se dividió en tres etapas: la primera de ellas consistió en el desarrollo de la tarjeta de mediciones propiamente; en la segunda, se abordó el problema de la digitalización de las variables adquiridas por la tarjeta diseñada y su debido acondicionamiento; y por último en la tercera etapa se abordó el problema de tomar los valores digitalizados y enviarlos a través del puerto USB. Para implementar esta última etapa se utilizó en circuito integrado FX2LP CY7C68013A, un circuito programable que permite la implementación de dispositivos que cumplan con el estándar USB 2.0.

De estas etapas la que presentó mayores dificultades fue la etapa de transmisión de datos, principalmente debido a la falta de recursos apropiados para la depuración del sistema, así como falta de conocimiento acerca de los requerimientos de energía del sistema de envío de datos; todo ello incidió en que al final no se pudieran alcanzar todos los objetivos del proyecto, en especial el de poder mandar correctamente a la computadora los datos obtenidos por la tarjeta desarrollada.

A pesar de que en esta investigación no se pudo lograr el objetivo final de enviar los datos por medio del puerto USB, el presente trabajo se puede concebir como un análisis de factibilidad para la implementación de sistemas similares al aquí expuesto.

1. CAPÍTULO 1: Introducción.

En los últimos años se ha hecho bastante común el uso de la instrumentación virtual, ya que permite obtener herramientas de medición y control a precios relativamente bajos y sobre todo simplificando el trabajo del usuario final. Sin embargo muchas de estas herramientas están diseñadas como módulos de expansión del ordenador (p.e. tarjetas PCIA) que hacen que la herramienta sea dependiente de un determinado tipo de arquitectura de computadoras.

El problema principal de este tipo de módulos radica en que se debe ser capaz de transmitir grandes cantidades de información en tiempo real, lo que obliga a utilizar puertos de comunicación que sean capaces de transmitir y recibir tales datos a la velocidad apropiada. Hasta hace algún tiempo esto solo era posible con buses internos al ordenador, es decir, por medio de circuitos conectados directamente a la tarjeta principal y mapeados directamente a la memoria principal, sin embargo con el desarrollo del puerto USB y principalmente con el desarrollo de su versión 2.0, que permite alcanzar un ancho de banda máximo teórico de 480 Megabits por segundo, se abre la posibilidad de desarrollar nuevos circuitos que puedan transmitir grandes cantidades de información, que tengan conexión y desconexión en caliente, es decir, con el computador encendido, y sobre todo que sean independientes de la arquitectura.

El presente trabajo trata sobre el diseño e implementación de un circuito destinado a hacer mediciones de diversas variables eléctricas básicas de manera que se pueda comunicar con el ordenador y enviar los datos obtenidos utilizando el puerto USB en su versión 2.0. Dada la complejidad del mismo, este proyecto solo trata sobre el desarrollo del

hardware y el firmware de la tarjeta, así como el controlador que se debe utilizar en el ordenador para poder hacer las transacciones de información de manera adecuada. Se dejará para posteriores trabajos el desarrollo de la interfaz gráfica para el usuario y otras funciones de control más complejas.

1.1 Objetivos.

Objetivo general.

- Diseño de una tarjeta de adquisición de datos para medición de variables eléctricas.

1.1.2 Objetivos específicos.

- Diseñar e implementar una "tarjeta de adquisición de datos" que permita realizar mediciones de diferentes variables eléctricas con una precisión razonable.
- Convertir los valores obtenidos por el circuito en señales eléctricas propias del puerto USB.
- Utilizar el puerto USB para adquirir los datos proporcionados por un circuito medidor de variables eléctricas y almacenarlos para su posterior utilización.

1.2 Metodología.

Dada la complejidad del trabajo se optó por utilizar una metodología de trabajo modular, es decir, dividiendo el proyecto en pequeñas partes; de esta manera se asegura que en caso de fallo sea más fácil detectar su causa.

Primeramente se debe hacer un estudio sobre el bus serie universal o USB por sus siglas en inglés, en su versión 2.0, a fin de identificar sus características eléctricas, tasas de

transferencia y protocolos. Con esto se puede determinar la frecuencia máxima de operación y la precisión del circuito medidor que se desea diseñar. Luego se debe seleccionar un circuito integrado que pueda realizar las funciones de un dispositivo USB, se debe poner atención en esta parte, ya que la programación del bus USB no es sencilla, así que para minimizar los tiempos de desarrollo se debe seleccionar un componente que tenga suficientes herramientas y librerías que permitan programar las partes esenciales del protocolo de una manera fácil o si es posible automática. También, se debe considerar que existen en el mercado circuitos con microprocesadores embebidos que permitirían controlar el circuito de mediciones, reduciendo el número de componentes a interconectar, facilitando la tarea de depuración del circuito.

Una vez seleccionado el integrado que proporcionará la interfaz USB, se desarrollará un circuito de medición de variables eléctricas básicas, como corriente, voltajes, resistencia, capacitancia y frecuencia. Se debe señalar que inicialmente el circuito no pretende ser utilizado como equipo de precisión, por lo tanto se utilizarán circuitos conocidos que permitan recolectar las magnitudes requeridas de una manera bastante sencilla, pero asegurando tanto una precisión como una respuesta en frecuencia aceptables. Es de fundamental importancia la selección del convertidor analógico-digital, ya que este definirá tanto la precisión como la frecuencia máxima de operación. Es en este punto donde se debe tener en cuenta las limitaciones del bus USB respecto a las tasas de transferencia máximas reales, que limitan el número de muestras por segundo que se pueden tomar, así como la exactitud de las mismas. También se debe tener cuidado de implementar los circuitos de sobrecorriente y sobrevoltaje necesarios para proteger tanto el circuito en sí

como las computadoras que se van a conectar a él. Para simplificar el diseño del circuito, también se dividirá este en submódulos.

Uno de los problemas con los que habrá que lidiar es el hecho de que, debido al constante avance en los empaquetados de circuitos integrados, gran parte de los componentes actuales solo se pueden conseguir en versiones con empaquetados para montura superficial, lo que dificulta la elaboración de prototipos.

Por último se debe escribir el firmware que controle tanto el circuito de medición como la transferencia de los datos hacia el ordenador, para esto se utilizarán herramientas de uso gratuito bajo el sistema operativo Windows.

CAPÍTULO 2: Desarrollo teórico.

2.1 Sistemas de instrumentación digital.

En la ciencia e ingeniería siempre ha existido la necesidad de realizar mediciones de diversas magnitudes con el fin de adquirir datos reales sobre el comportamiento de algún fenómeno físico, corroborar los resultados teóricos con datos reales o comprobar el correcto funcionamiento de los equipos y maquinarias tanto de laboratorio como industriales. Esta necesidad de medidas hizo aparecer multitud de equipos destinados a la medición de las más diversas variables físicas, sin embargo, la mayor parte de estos equipos de medición se basa en el uso de transductores que transforman una variable física de cualquier índole en una variable eléctrica, esto debido a que las variables eléctricas son más fáciles de manipular manteniendo una precisión que en algunos casos puede llegar a ser muy alta. Los primeros equipos instrumentales electrónicos que aparecieron eran de tipo analógico. Por lo general estos disponían de una aguja de cuadro móvil para desplegar los valores medidos, la cual presenta problemas como no linealidad del elemento de desviación y el fenómeno de paralaje. Otros disponían de una pantalla electro-fosforescente como los osciloscopios.

Conforme la técnica de fabricación de componentes electrónicos avanzó, aparecieron dispositivos de medición en los cuales se utiliza una pantalla digital como medio para desplegar los datos obtenidos. Estos instrumentos se basan en la conversión de una señal analógica, que puede tomar cualquier valor real, en una digital, en la cual los valores solo pueden tomar un valor discreto, con lo que se evitan los problemas de no linealidad de los componentes, fuentes de ruido, etc, que pueden afectar la precisión de los datos obtenidos, además de evitar el problema del paralaje. Sin embargo, estos primeros

medidores digitales tenían un ancho de banda bastante reducido, un par de miles de hertz, por lo que no desplazaron por completo a los dispositivos de tipo analógico, en especial aquellos que tenían anchos de banda bastante más amplios, como los osciloscopios. En los últimos 20 años las técnicas de conversión analógica a digital se han perfeccionado bastante con lo que los equipos digitales han aumentado considerablemente sus características de respuesta en frecuencia.

Actualmente se puede conseguir en el mercado dispositivos para realizar la conversión de una señal analógica a digital que pueden muestrear señales a frecuencias superiores a 1 GHz. El creciente aumento en la velocidad de estos equipos trae consigo la necesidad de utilizar componentes electrónicos de alta frecuencia, que son sustancialmente más caros que sus homólogos de baja frecuencia, y a poner especial atención al diseños de la placa de circuito impreso, PCB, a fin de evitar problemas tales como resonancia y "rebote" de señales en las vías del PCB, así como los fenómenos de interferencia electromagnética, IEM, y otros relacionados con las altas frecuencias. Todos estos factores contribuyen a que el costo final del equipo sea considerablemente alto.

Sin embargo en los últimos años han comenzado a aparecer sistemas de instrumentación basados en el uso de las computadoras electrónicas. Debido a que actualmente las computadoras pueden alcanzar altas velocidades de cálculo a un precio razonablemente bajo, se evita el uso de componentes discretos especializados transfiriendo las tareas de estos al procesador de la computadora y al software de la misma, con lo que el precio final del sistema se ve reducido sustancialmente. La mayoría de estos sistemas están diseñados para trabajar en computadoras personales PC, como tarjetas internas conectadas

al puerto PCIMA. Esto presenta un problema, ya que el dispositivo no se puede utilizar en computadoras con una arquitectura diferente.

2.2 Convertidores analógico-digital.

2.2.1 El CAD.

El convertidor analógico digital, CAD, es uno de los bloques más importantes en los sistemas de medición digital. Como su nombre lo indica, se encarga de transformar una señal analógica en una señal digital equivalente. Para hacer esta conversión, el CAD compara la señal de entrada contra un valor de referencia, y expresa esta como una fracción de la referencia. Es decir, el CAD es básicamente un divisor y la ecuación que relaciona la entrada con la salida para un conversor analógico digital es

$$V_{Sal} = \left\lceil ٢^{n} G \frac{V_{Ent}}{V_{ref}} \right\rceil \qquad (2.2\text{-}1)$$

donde el símbolo $\lceil \ \rceil$ representa el entero inmediato superior

“n” es el número bits de salida del CAD, resolución

“G” es la ganancia del conversor, usualmente es 1

“Vent” es el voltaje analógico de entrada

“Vref” es el voltaje de referencia del conversor

El valor de “n”, el número bits que utiliza el CAD para representar el voltaje de entrada, también se llama resolución, ya que indica cual es el menor valor de voltaje que el conversor puede representar, este valor está directamente relacionado con el valor del error de cuantización.

El error de cuantización “q” indica cuál es la máxima incertidumbre que se puede tener a la hora de convertir una señal analógica en digital. Por lo general “q” es igual a un medio del bit menos significativo de la salida del conversor. Es fácil ver que cuanto mayor sea el número de bits con que se representa la señal, menor será “q” y por tanto el instrumento de medida tendrá una mayor precisión.

La figura 2.2.1 muestra una gráfica de voltaje de entrada vs voltaje de salida y su error de cuantización para un CAD con una resolución de tres bits

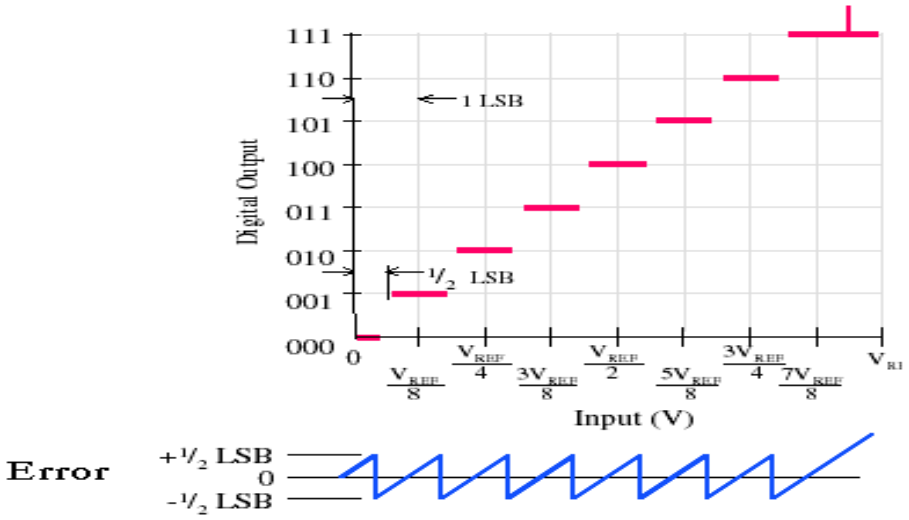


Figura 2.2.1 Error de cuantización de un CAD ideal.

Otro aspecto importante que afecta a los CAD es la relación señal-ruido RSR, que para un CAD ideal es

$$RSR = 20\log\left(2^{n-1}\sqrt{6}\right) \qquad (2.2\text{-}2)$$

Por otra parte, dado que los CAD son circuitos estrictamente no lineales, al aplicar una señal senoidal pura a la entrada se obtendrá a la salida la señal senoidal más sus

armónicas. Para relacionar la potencia de las armónicas en términos de la fundamental se utiliza la distorsión armónica total DAT, la cual está definida por

$$DAT = \sqrt{\frac{V_{f1}^2 + V_{f2}^2 + ... + V_{fn}^2}{V_f^2}} \qquad (2.2\text{-}3)$$

Donde V_{fi} es el valor pico de la i-enésima armónica y Vf es el valor pico de la fundamental. Normalmente se expresa en porcentaje o en dB.

El SRS y el DAT afectan negativamente la salida del CAD haciendo que se pierdan ciertos códigos de salida, ya que el efecto combinado de ambos hace que el valor de "leído" por el conversor nunca esté en el rango de valores para ese código. La figura 2.2.2 muestra un ejemplo de códigos perdidos para un CAD de tres bits.

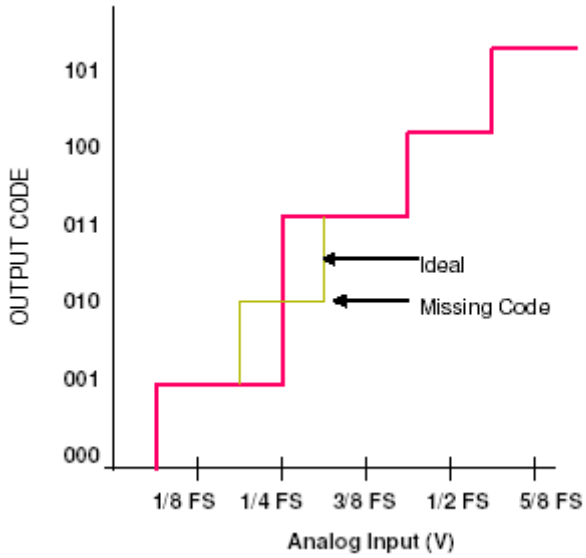


Figura 2.2.2 Códigos perdidos.

Estos códigos perdidos hacen que el conversor se comporte como si fuese uno ideal con una resolución menor. El número efectivo de códigos de salida se calcula como

$$NEDB = \frac{-20\log\sqrt{10^{-RSR/10} + 10^{DAT/10}} - 1.76}{6.02} \qquad (2.2\text{-}4)$$

Otro aspecto que se debe tomar en cuenta a la hora de usar CAD es la estabilidad del reloj; es necesario que este sea bastante estable, ya que si presenta variaciones en su periodo, frecuencia o ciclo de trabajo, puede provocar severas distorsiones a la señal de salida. También se debe poner especial atención a las fuentes generadoras de voltajes de referencia, ya que este debe ser lo más constante y preciso posible.

2.3 Multímetro digital.

La medición de magnitudes tales como voltaje, corriente, resistencia, capacitancia, inductancia y frecuencia son muy importantes en los sistemas eléctricos. Un sistema capaz de medir estas y otras variables se llama multímetro o polímetro. Un multímetro debe ser capaz de medir tensiones y corriente tanto en DC como en AC y desplegar su valor absoluto o su valor medio cuadrático o RMS respectivamente. Además, dependiendo de las características del mismo, es capaz de mostrar información tal como la frecuencia de la señal medida, su valor pico, etc. El principio de operación de un multímetro, como en todo sistema de instrumentación, es convertir la variable a medir en una variable que sea más fácil de representar. En los sistemas digitales modernos esta variable es por lo general un voltaje que luego es transformado en un código binario apto para ser procesado por los sistemas de medida.

2.3.1 Medición de voltaje.

Idealmente un instrumento destinado a la medición de voltaje debe tener una impedancia infinita, a fin de asegurar que el proceso de medida no afecte el sistema que se

desea observar, y por tanto no modifique la variable observada. Por esto, es deseable que el sistema de medición de voltajes tenga una impedancia lo más alta posible. Existen diversos métodos para proporcionar una impedancia de entrada alta que pueden proporcionar impedancias en el orden de los teraohm ($10^{12}\ \Omega$) también llamados electrómetros. Entre ellos está utilizar amplificadores con entradas que tengan transistores de efecto de campo (FET por sus siglas en ingles), en forma diferencial o entradas de acople capacitivo, que utilizan una placa de metal como elemento sensor. Estos métodos de sensado por lo general tienen una mala respuesta en frecuencia y su uso queda restringido a aplicaciones en las que el sistema tiene una impedancia de salida muy alta y el voltaje o campo eléctrico cambia en forma muy lenta, tal es el caso de los medidores de PH y ciertos microscopios electrónicos. Otra forma más barata y que brinda resultados relativamente buenos es utilizar un simple divisor de voltaje tal y como se muestra en la figura 2.3.1.

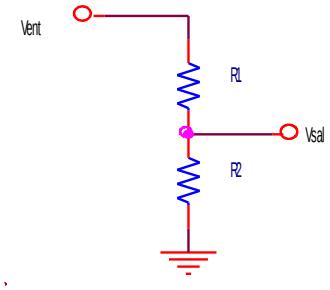


Figura 2.3.1. Divisor de voltaje.

El voltaje de salida se relaciona con el voltaje de entrada mediante la fórmula

$$V_{Sal} = \frac{R2}{R1+R2} V_{Ent} \qquad (2.3\text{-}1)$$

Donde Vent es el voltaje que se desea medir.

Vsal es el voltaje de salida.

Este método para acondicionar el voltaje por medir es válido solo cuando el voltaje de entrada es mayor que el máximo voltaje que puede soportar el CAD.

En caso de que el voltaje de entrada sea considerablemente menor al máximo valor que el CAD puede soportar, es conveniente utilizar un amplificador a fin de obtener una mejor precisión.

Se debe prestar atención a las resistencias de sensado. En general no es fácil dar una regla para la selección de estas, ya que se debe tener un compromiso entre una impedancia suficientemente alta, como para no cargar mucho el circuito en el cual se quiere hacer la medición, y una lo suficientemente pequeña como para que no se amplifique demasiado el ruido térmico que siempre va a estar presente en cualquier componente electrónico; además se debe tener en cuenta la tecnología de las resistencias, ya que esta determina no solo su sensibilidad al ruido, sino que define la máxima frecuencia a la que pueden operar, pero tampoco se debe dejar de lado el compromiso económico. Se puede decir que para resistencias de carbón de 1% de error, una impedancia de entrada de un par de magaohms es suficiente.

2.3.2 Medición de corriente.

Dado que los CAD por lo general funcionan transformando valores de voltaje a código binario, es preciso poder convertir las magnitudes de corriente a una magnitud de voltaje equivalente. Nuevamente la manera más fácil es utilizar una resistencia de valor

conocido y por la ley de Ohm se sabe que el voltaje instantáneo a través de una impedancia es el producto de la corriente instantánea por la impedancia (instantánea). Para el caso de resistencias comerciales el valor de la impedancia es prácticamente constante y real cuando se trabaja a bajas frecuencias, por lo que resulta un método bastante aceptable.

Esta resistencia de sensado debe ser lo más cercana a cero, lo cual presenta un serio reto, ya que si la resistencia es muy baja la diferencia de potencial que aparecerá en sus terminales será también muy baja, provocando que este se pueda confundir con el ruido térmico o perturbaciones externas, mientras que si es muy alto el circuito bajo medida se verá afectado considerablemente afectando la precisión de la medida. Si se quieren utilizar resistencias de bajo valor, lo más recomendable es que tanto esta como el circuito acondicionador de señal (amplificador) cuenten con un método para apantallar las perturbaciones electromagnéticas externas y que el circuito amplificador tenga una buena razón de rechazo de modo común, esto se traduce en la utilización de amplificadores de instrumentación y colocando todo el conjunto dentro de una caja metálica conectada al punto de referencia del circuito de medida. Para diseños de alta frecuencia se debe poner atención al diseño de esta caja metálica, ya que su presencia introduce una capacitancia parásita que a su vez provoca lazos de realimentación indeseables.

Por otra parte, cuando se desea medir corrientes de magnitud relativamente elevada, se debe poner atención a la potencia disipada a través de la resistencia de sensado, ya que si su temperatura se eleva por encima de su temperatura de operación máxima podría variar su valor óhmico o dañarse por completo.

2.3.3 Medición de resistencias.

Existen varios métodos para la medición de resistencias, los más precisos consisten en la utilización de puentes de resistencias de precisión con un valor conocido y colocar la resistencia de valor desconocido en uno de los brazos; sin embargo estos métodos son bastante tediosos y generalmente no se necesita conocer el valor de aquellas con mucha precisión, p.e., para un diseño de circuitos electrónicos un error de 1% es aceptable.

Para efectuar mediciones del valor óhmico de las resistencias se recurre nuevamente a la ley de Ohm. En este caso para convertir la magnitud de la resistencia en un voltaje equivalente se hace pasar una corriente de valor constante conocida a través de la resistencia desconocida y se mide la diferencia de potencial en sus extremos. Luego con una sencilla relación matemática se puede saber el valor de la misma.

El método anteriormente descrito puede ser utilizado para medir resistencias con valores comprendidos entre unos cuantos ohmios hasta varios megaohmios. Si se desea poder medir valores óhmicos inferiores a la unidad, se debe utilizar corrientes bastante elevadas para poder obtener un voltaje perceptible en sus extremos, nuevamente se debe tener cuidado que la resistencia no llegue a valores de temperatura peligrosos debido a una corriente demasiado grande a través de ella.

2.3.4 Medidas de capacitancia.

Para la medición de capacitancias también existen varios métodos, que si bien dan valores bastante exactos, su uso queda restringido al interior de laboratorios. Una forma sencilla de encontrar la capacitancia consiste en aprovechar el comportamiento de los

sistemas de primer orden para obtener una medida bastante aproximada de las capacitancias.

Un circuito de primer orden tiene una ecuación de la siguiente forma

$$V_{Sal} = (1 - e^{-t/\tau}) \qquad (2.3\text{-}2)$$

donde τ es la constante de tiempo del sistema.

En el caso de redes de una resistencia y un capacitor, el valor de esta constante es igual al producto del valor óhmico por el valor de la capacitancia en faradios. Si el valor de la resistencia es conocido, es posible determinar el valor de la resistencia a través de la medición del tiempo de carga o descarga del capacitor.

En la práctica, en lugar de medir el tiempo de carga o descarga del sistema, este se lleva a una condición de metaestabilidad; en este punto la frecuencia de oscilación del sistema es un valor proporcional a la constante de tiempo, por lo que se puede utilizar un frecuencímetro para obtener una medición indirecta de la capacitancia.

2.3.5 Medidas de frecuencia.

Dentro de la ingeniería eléctrica muchas veces es necesario poder realizar mediciones de valores de frecuencias relativamente altas, para esto se utiliza el frecuencímetro.

El principio de operación de un frecuencímetro se basa en contar el número de veces que la señal por medir pasa por el mismo valor de voltaje o corriente durante un tiempo determinado, luego se divide la cantidad de pulsos contada por la base de tiempo utilizada.

CAPÍTULO 3: Diseño del circuito de adquisición.

El sistema que se describe en este proyecto se basa en el uso de una tarjeta de adquisición de datos externa al computador y que se conecta al mismo por medio del puerto USB en su versión 2.0; con esto se evita la necesidad de tener que abrir la computadora para extraer el equipo de adquisición y además presenta la ventaja de que el puerto USB se utiliza en la mayor parte de las computadoras actuales por lo que el sistema es multiplataforma. Este proyecto no pretende innovar en este campo, ya que el autor conoce la existencia de sistemas de adquisición similares basados en el estándar USB, sin embargo, la mayoría de estos se basan en versiones antiguas del estándar, por lo que el sistema aquí propuesto puede alcanzar en teoría velocidades de transferencia de datos mucho mayores que la mayoría de estos.

3.1 Diseño del circuito de adquisición.

Para la selección del CAD y los componentes necesarios para su funcionamiento se deben tomar en cuenta varios factores como la respuesta en frecuencia de los amplificadores por utilizar, el voltaje mínimo que se pretende medir y el voltaje de entrada del CAD, si el CAD necesita una referencia externa, así como la precisión deseada.

Tomando en cuenta los datos de diseño se seleccionó el CAD AD7821TE con arquitectura half-flash fabricado por Analog Devices. Este tiene una velocidad de muestreo de hasta 1 Mmps y puede soportar voltajes de entrada entre 0 V y 5 V. El AD7821 está diseñado para ser utilizado con microcontroladores, por lo que su interfaz es bastante simple, además, al ser del tipo half-flash no necesita de un reloj externo. Este integrado está dotado de un circuito interno de retención de valores a la entrada (track-and-hold) que

permite muestrear señales de hasta 100 kHz, por lo que, siempre y cuando no se necesite muestrear señales de más de 100 kHz, no es necesario utilizar un circuito de retención externo. Nótese que cuando el CAD funciona a su máxima velocidad de adquicisión puede tomar como mínimo diez muestras a la señal de entrada, de esta forma, se disponen de bastantes puntos de medición que faciliten al software que ha de procesar los datos, una reconstrucción fidedigna de la onda de entrada.

Como en todo sistema de conversión de señales analógicas a discretas, se necesita limitar el ancho de banda de las señales que llegan al convertidor, para esto se utilizó un filtro analógico tipo Butterworth de 4 orden cuya frecuencia de corte está en los 100 kHz y se implementó utilizando celdas Sallen-Key.

Para la etapa de amplificación se utiliza un INA2331, que es un amplificador de instrumentación doble con una ganancia mínima de 5 V/V. Este circuito tiene un ancho de banda superior a los 300 kHz cuando la ganancia es igual o menor a 100 V/V, por lo cual es apto para esta aplicación. La figura 3.1.1 muestra los componentes que conforman el INA2331

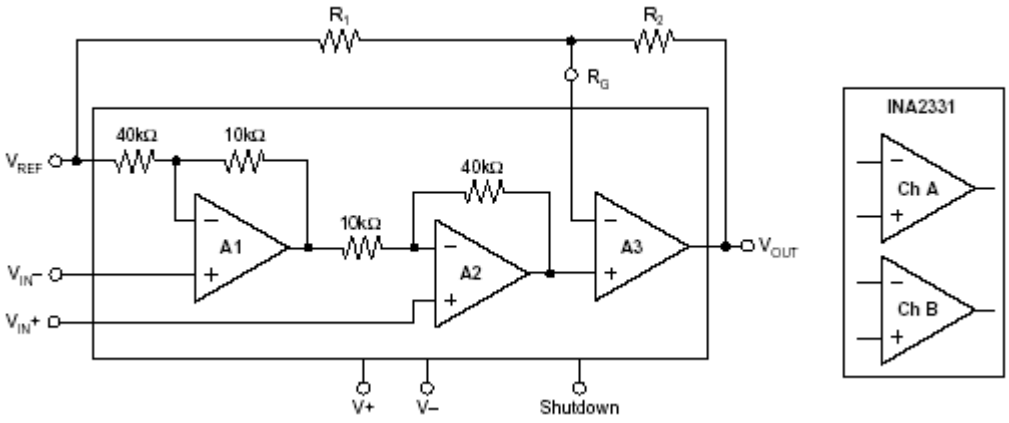


Figura 3.1.1 Diagrama del INA2331.

Dado que el circuito debe medir señales tanto continuas como alternas, y el CAD solo puede convertir señales positivas, es preciso dotar al circuito de un sistema para poder medir los valores negativos de las señales. Existen varias formas de hacer esto y en este proyecto se escogió utilizar un amplificador operacional en serie con el amplificador de instrumentación de tal forma que el AO sume un voltaje de 2,5 V a la salida del AI, de esta forma se evita tener voltajes negativos en la entrada del CAD, además el AO sirve para acondicionar la señal a los valores de impedancia del CAD.

3.2 Diseño del voltímetro.

Una vez diseñado el circuito de adquisición de datos solo basta escoger los valores de las resistencias que conforman el divisor de tensión.

De las especificaciones de diseño se tiene que el circuito medidor de voltaje debe ser capaz de medir voltajes entre 0.5 V y 500 V, además se debe tomar en cuenta que el voltaje máximo a la entrada del amplificador de instrumentación no debe ser superior a los 5 V; por esto, para voltajes de entrada mayores de 5 V se debe pasar este voltaje a través de un divisor de tensión adecuado. Para tal fin se dispone de varios switches que permiten escoger entre diversas fracciones del voltaje de entrada.

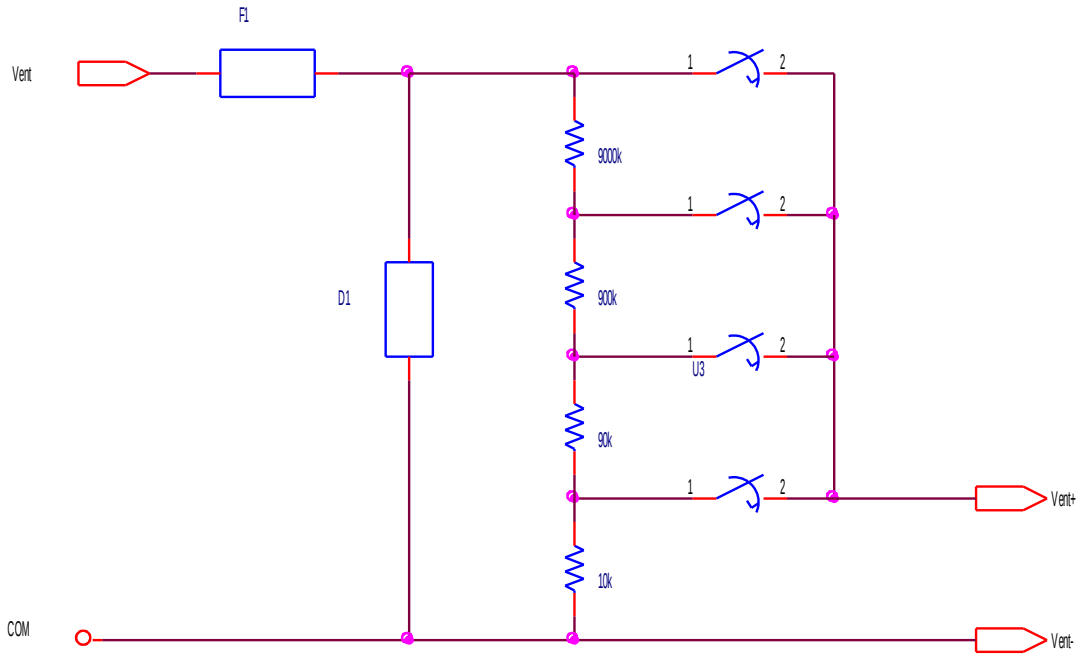


Figura 3.2.1 Circuito acondicionador para el voltímetro.

En la figura 3.2.1 se puede ver el circuito acondicionador de señal del voltímetro. El elemento marcado como D1 es un diodo varactor de 500 V y F1 es un fusible de 100 µA X 100 V

Para el caso del voltímetro, se selecciona la ganancia del AI en 5 V/V

3.3 Diseño del amperímetro.

Como ya se mencionó, un amperímetro debe tener una impedancia de entrada lo más cercana posible a cero. En la práctica esta impedancia está limitada por un valor de la resistencia que se utiliza para obtener un voltaje equivalente a la corriente de muestra. Se aprecia como necesario que el voltaje máximo que se producirá a plena carga debe ser lo más pequeño posible. A pesar de que el amplificador de instrumentación que se usa puede

proporcionar ganancias de hasta 500 V/V, no se puede usar una ganancia muy elevada, ya que esto reduce el ancho de banda del amplificador y por tanto del sistema. Por esto se escogió que la ganancia del AI se fijara en 150 V/V, con lo que se obtiene un voltaje de 0,0167 V a plena escala.

El amperímetro diseñado tiene tres escalas de medición que permiten medir corrientes tanto negativas como positivas de hasta 500 mA con una precisión teórica de hasta 3 pA. Pese a que el voltaje de entrada del AI es bastante pequeño, se tienen impedancias de entrada de hasta 330 Ω que pueden llegar a ser intolerables en algunos sistemas sensibles. En la tabla 3.1.1 se observan los rangos de corriente que se pueden medir y el valor de la resistencia utilizada.

Tabla 3.3.1 Rangos de corriente del amperímetro.

Para los dos últimos rangos de corriente se observa que las resistencias necesarias para producir el voltaje de plena carga son bastante bajas y no existe ningún fabricante conocido hasta el momento que produzca tales valores, por lo que estas se deben fabricar a partir de materiales disponibles en el laboratorio.

Para la fabricación de la resistencia de 0,03 Ω se utilizó una placa de circuito impreso. La cantidad de cobre utilizada para fabricar las placas de circuito impreso suele expresarse en onzas/pie^2; para una placa de 1 onz/pie^2 (30 g/m^2) se tiene que el espesor de la capa de cobre es aproximadamente 0,00343 cm y tomando la resistividad del cobre como

1,822 $\mu\Omega$ /cm, se tiene una resistencia de 530 $\mu\Omega$ /unidad de área, de tal manera se necesitan aproximadamente 57 unidades de área de circuito impreso para obtener una resistencia de 0,03 Ω. Nótese que el resultado es independiente del ancho de la pista creada, de hecho el ancho de la pista solo está limitado por la capacidad de corriente que puede soportar el circuito sin dañarse; usualmente por cada milímetro de ancho de la pista pueden pasar 2 A sin que esta se caliente de forma apreciable, de esta forma se selecciona una pista de 3 mm de ancho por 171 mm de largo, para que soporte la máxima corriente que se debe medir.

Para la selección de las diferentes resistencias se utilizan relevadores, ya que la mayoría de los circuitos integrados de conmutadores analógicos tienen resistencias de encendido mucho mayores a 0.1 Ω que es considerablemente mayor a 0,03 Ω.

Una vez que la señal es amplificada pasa por el desplazador de nivel y luego se envía al canal B del CAD a fin de poder medir el voltaje y la corriente de forma simultánea.

3.4 Diseño del ohmímetro.

En la figura 3.4.1 se observa el diagrama esquemático de un circuito convertidor de voltaje a corriente. Si en este circuito el voltaje V_{ent} se mantiene constante, entonces la corriente I_L también será constante. La corriente I_L se puede calcular con la siguiente fórmula:

$$I_L = \frac{V_{cc} - V_{ent}}{R} \qquad (3.4\text{-}1)$$

La corriente máxima que se puede obtener de este circuito es $I_L = I_{AO} * h_{FE}$, donde I_{AO} es la corriente máxima que puede proporcionar el amplificador operacional y h_{FE} es la

ganancia del transistor en CD. Se debe prestar atención a la corriente de corte en el colector del transistor, ya que esta define cuál será la máxima resistencia que se puede medir con el circuito. También se debe tener cuidado que el voltaje en la resistencia que se está probando no supere el voltaje de saturación del amplificador.

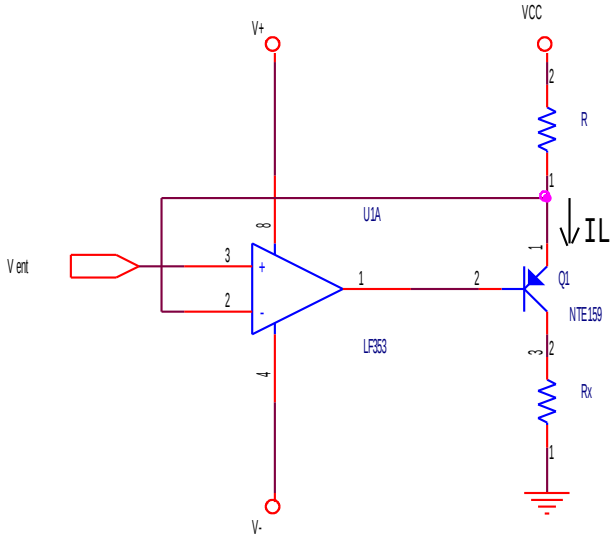


Figura 3.4.1 Fuente de corriente constante.

Tabla 3.4.1 Valores de resistencias para las escalas del ohmímetro.

3.5 Diseño del frecuencímetro.

En esta sección se trata el diseño del frecuencímetro. El circuito aquí mostrado puede operar hasta una frecuencia aproximada de 50 MHz. El circuito medidor de frecuencias consta de dos partes principales:

1. Un contador de pulsos con reposición.
2. Un acondicionador de señal, que modifica la señal de entrada a fin de poder aplicarla al contador.

En la figura 3.5.1 se observa el circuito del acondicionador de señal. El transistor NTE312 es un transistor de efecto de campo de canal N para altas frecuencias. Este se encarga de proporcionar una entrada de alta impedancia al circuito. Los diodos 1N914 sirven como limitadores de voltaje a fin de proteger el amplificador. Por otra parte el transistor 2N3646 es un transistor de conmutación para alta frecuencia tipo NPN y se encarga de transformar la forma de onda entrante en una onda cuadrada. Por último se cuenta con un circuito tipo 74HC04 que se encarga de acondicionar los niveles de voltaje de entrada a niveles lógicos aptos para la familia 74HC.

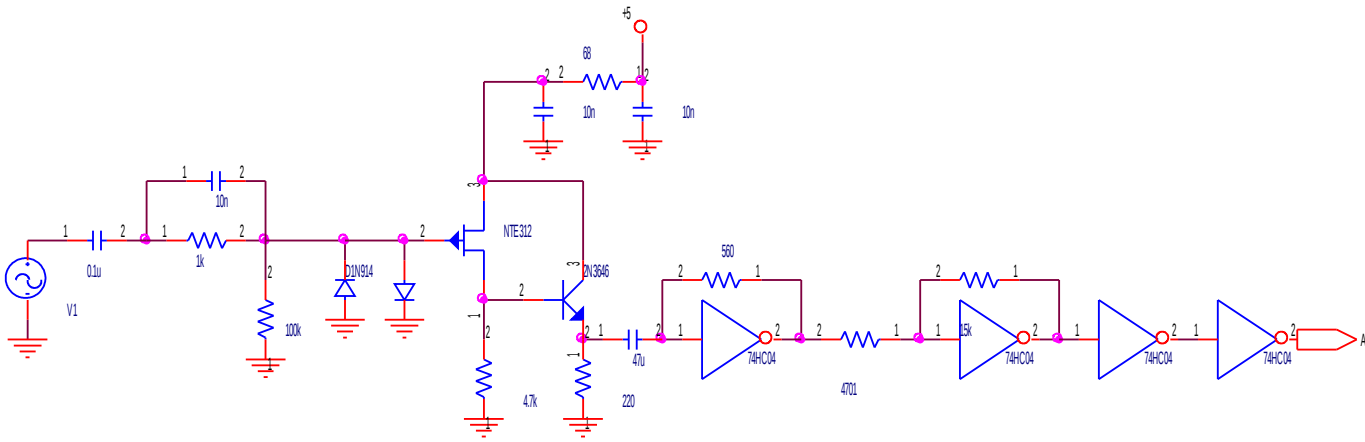


Figura 3.5.1 Acondicionador de señal para el circuito contador.

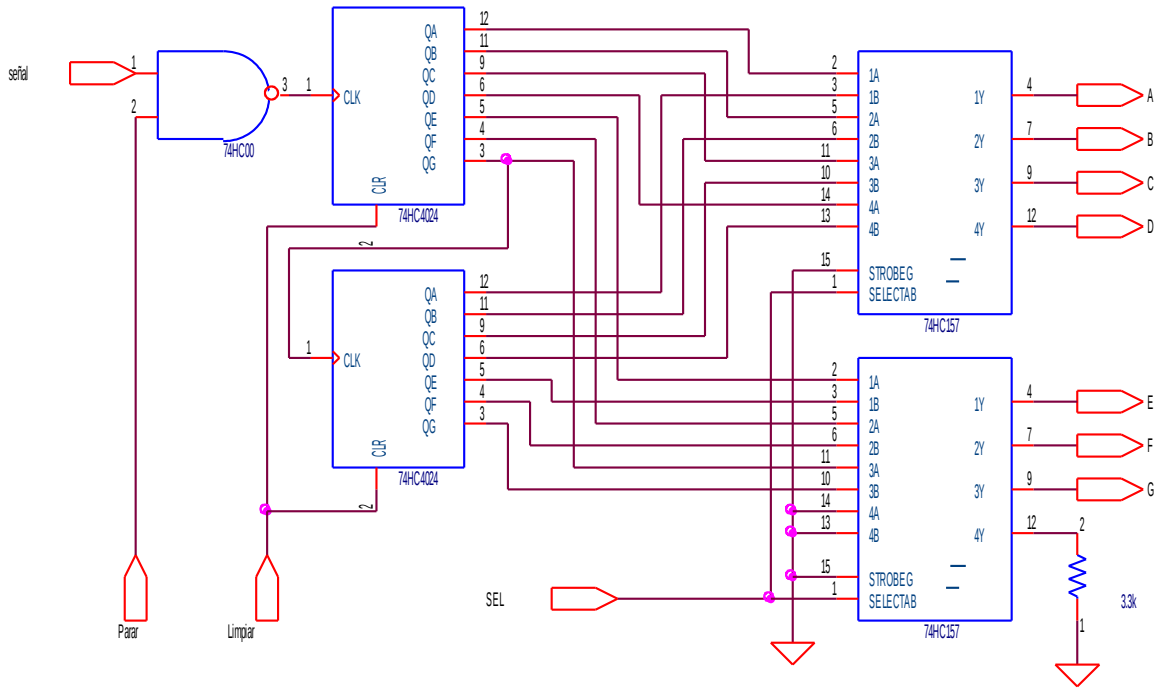


Figura 3.5.2 Circuito contador.

El circuito contador está formado por 2 CI tipo 74HC4024, que es un contador binario de 7 bits con reposición asincrónica conectados en serie, por lo que el circuito puede medir frecuencias hasta con cuatro dígitos de precisión. Dado que el transceptor USB solo tiene una entrada de ocho bits en paralelo, las salidas de los contadores están multiplexadas.

El funcionamiento del contador es el siguiente: la señal que se desea medir se hace pasar por el acondicionador de señal y llega a la compuerta NOY que en su otra entrada tiene conectada la señal de parada que viene del controlador. Si la señal está en alto la compuerta se comporta como un inversor dejando pasar la señal, invertida, hasta el contador. Una vez que pasa un tiempo determinado por el controlador, la señal de parar se

pone en bajo inhabilitando el paso de la señal medida, luego por medio de la señal SEL se selecciona la parte baja del dato obtenido y se envía al transceptor, luego se toma la parte alta y se reestablece el circuito poniendo la señal Limpiar en alto, para repetir el proceso.

3.6 Diseño del capacitímetro.

Para el diseño del capacitímetro, como se explicó anteriormente, se hará un circuito que oscile a una frecuencia proporcional al valor de la capacitancia a medir. Para esto se utilizará un cd4047 funcionando en modo astable como se muestra en la figura 3.6.1

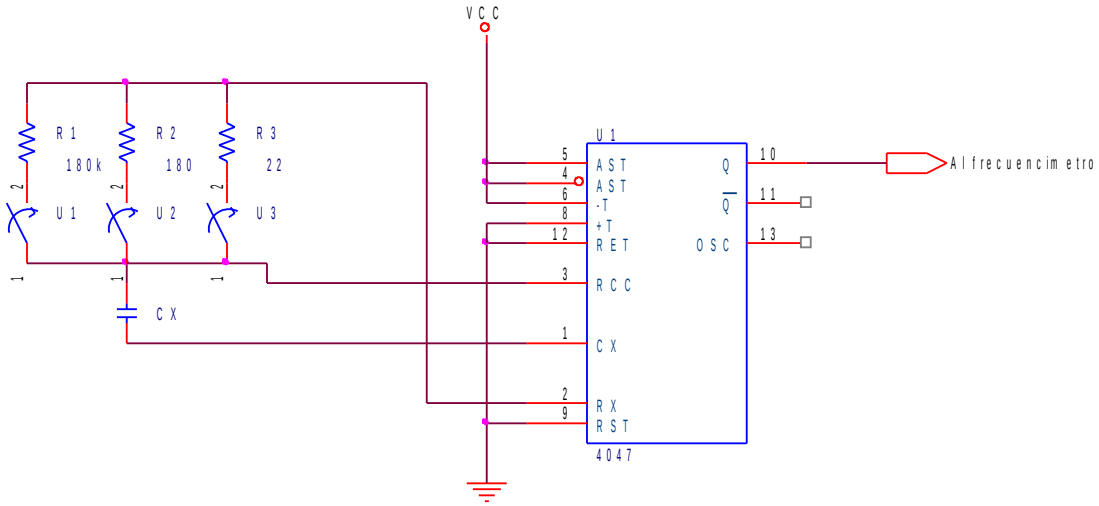


Figura 3.6.1 Circuito para la medida de capacitancias

El circuito debe ser capaz de medir capacitancias entre 1 pF y 1 mF con un periodo de muestreo no mayor a 1, a fin de que el usuario no deba esperar demasiado para obtener un valor fiable. Esta limitación en el tiempo de muestro hace que la precisión para capacitancias relativamente altas disminuya considerablemente, ya que en el peor de los casos el circuito oscilará a una frecuencia de 10 Hz.

La frecuencia de oscilación de este circuito está dada por la fórmula

$$f = \frac{1}{4.4 * R * C} \qquad (3.6\text{-}1)$$

Donde R está dado en ohmios y C en faradios.

El CD4047 en teoría puede operar a frecuencias de hasta 10 MHz, pero a estas frecuencias la precisión de la frecuencia no es muy buena por lo que se decidió restringir la frecuencia máxima de operación a 1.2 MHz.

Con estos datos se seleccionaron los valores de las resistencias del circuito. Para las resistencias R1 y R2 se definió una frecuencia de operación mínima de 1 kHz a fin de mantener una buena precisión. Estas resistencias permiten medir capacitancias con rangos entre 1 pF y 1 nF para la resistencia R1; 1nF y 1 μF cuando se selecciona la resistencia R2. Por otra parte para la resistencia R3 se seleccionó una frecuencia de operación mínima de 10 Hz a fin de que cuando se seleccione esta resistencia, el circuito pueda medir capacitancias entre 1 μF y 1 mF

3.7 Diseño de la interfaz del circuito analógico con el módulo USB.

En el desarrollo de este proyecto se probaron varias alternativas para interconectar la tarjeta de adquisición con el integrado USB.

Uno de los principales problemas afrontados fue el resultado de la tendencia de los fabricantes de circuitos integrados de ofrecer la mayoría de sus nuevos productos en encapsulados de montura superficial, que tienden a ser más baratos y facilitan su acomodo en tarjetas producidas a gran escala, pero dificulta el desarrollo de prototipos debido a que

no se pueden conectar directamente a una tarjeta de prototipos. Esto redujo en gran medida las opciones para seleccionar los integrados utilizados en el proyecto, ya que son pocos los fabricantes que producen integrados con anchos de banda relativamente altos en encapsulados DIL (Dual In Line) y aún menos las casas distribuidoras que los venden.

Otro problema de consideración fue el mismo FX2L. Como se mencionó anteriormente, para reducir el esfuerzo de diseño, se decidió utilizar el módulo de desarrollo PmodUSB2 de Digilent, el cual tiene solo cuatro pines dedicados a la interfaz de control, tres de salida y uno de entrada, y un bus de datos de ocho bits, además no cuenta con acceso a los pines de entrada y salida de reloj. Inicialmente se optó por soldar cables a los pines de entrada y salida de reloj del CY7C68013A, con el fin de poder trabajar con CAD que necesiten señales de temporización, pero esta aproximación no resultó óptima debido a la poca cantidad de señales de control disponibles en la tarjeta de desarrollo seleccionada. Ello obligó al uso de convertidores diseñados para microprocesadores, los cuales tienen un bus de datos de ocho bits y necesitan pocas señales de control, además muchos de estos chips trabajan con la tecnología Half-Flash que permite velocidades de muestreo relativamente altas sin la necesidad de un reloj externo, tal es el caso del AD7821TE.

CAPÍTULO 4: El estándar USB.

4.1 Introducción.

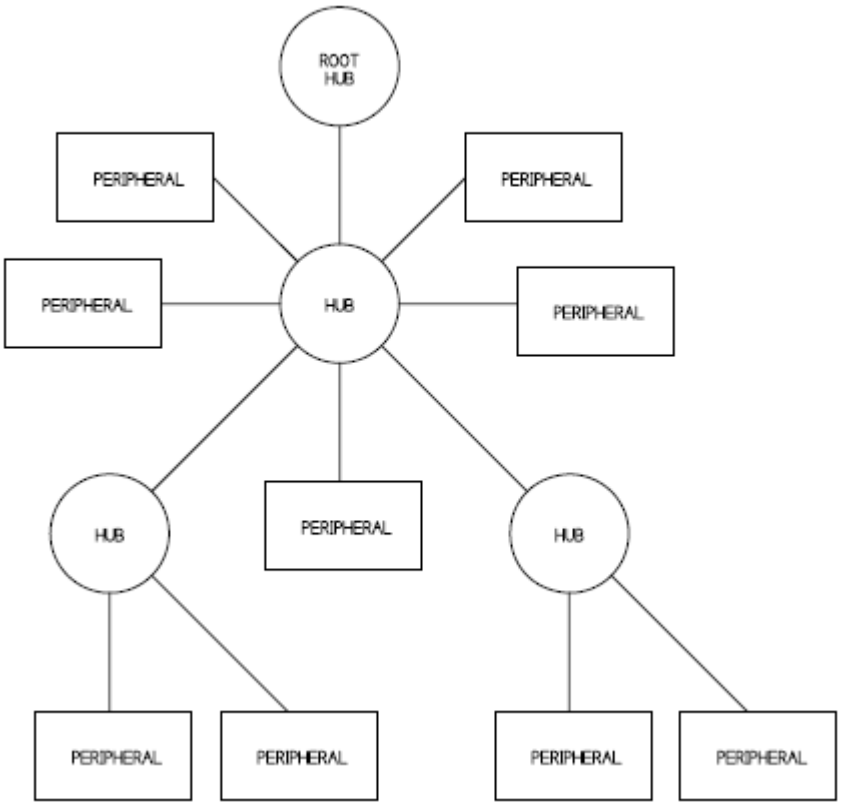
A continuación se describirá el estándar USB en su versión 2.0. Solo se dará una reseña de los puntos más importantes del mismo, si se requiere una descripción más detallada se debe referir a la descripción del estándar que se puede obtener de la página del mismo.

El bus serie universal, USB por sus siglas en inglés, fue creado a finales de 1996 por las empresas IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC con el fin de proporcionar una interfaz estándar para conectar diversos dispositivos al ordenador para facilitar su uso, conexión y configuración al usuario final. Esta primera versión permitía alcanzar velocidades máximas teóricas de hasta 12 Mbits por segundo, posteriormente en 2000 fue liberada la versión especificación que aumenta esta velocidad a 480 Mbits/s y que es totalmente compatible con la versión anterior.

Topológicamente hablando, el bus serie universal consiste en un solo servidor ("host") y múltiples dispositivos, llamados funciones, conectados en una estructura de árbol

por medio de concentradores ("hubs"). El "servidor" es una combinación de software y hardware, y se encarga entre otras cosas de detectar cuando se conecta o desconecta un dispositivo USB, así como configurarlo y cargar los controladores respectivos, iniciar y arbitrar todas las transferencias de información entre el ordenador y los dispositivos USB, administrar el consumo de energía, etc.

Las funciones de los concentradores son, entre otras, proporcionar más ranuras en las cuales se pueda conectar más dispositivos, y proporcionar energía, así como actuar de repetidor de los datos enviados desde y hacia el ordenador.



El estándar permite conectar a un servidor hasta 127 dispositivos, ya sean periféricos y/o concentradores. La máxima distancia del cable está determinada por las características eléctricas del cable, pero en general son aproximadamente 5 m, lo que permite alcanzar distancias relativamente grandes cuando se utilizan varios concentradores en cascada, sin embargo, debido a limitaciones tanto de software como de hardware, se recomienda limitar el número de concentradores conectados en cascada, ya que se puede impactar negativamente el ancho de banda real de los dispositivos conectados a dichos concentradores.

El estándar define 3 tipos de velocidades de transmisión de datos según las necesidades de los dispositivos, y a su vez, en cada velocidad se tienen diversos protocolos de intercambio de información, lo que proporciona gran flexibilidad a los diseñadores de hardware.

4.2 Arquitectura y terminología.

La arquitectura del un dispositivo USB genérico es multicapa. Un dispositivo consiste en una o más configuraciones, cada una de las cuales describe posibles formas de operación en las que el dispositivo puede ser programado, p.e. características de consumo de energía. Cada una de las configuraciones contiene una o más interfaces que son accesibles una vez se ha elegido una configuración en específico. Las interfaces proporcionan una definición de las funciones disponibles para el dispositivo.

A su vez cada interfaz contiene cero o más "endpoints", que son porciones del dispositivo USB que pueden funcionar como fuente de información o como destino de información.

La comunicación con el dispositivo se realiza por medio de un "pipe" que no es más que una asociación lógica entre un "endpoint" y un programa que se encuentra corriendo en el "servidor". En la práctica los términos "endpoint" y "pipe" son frecuentemente intercambiados.

4.3 Especificaciones eléctricas y de energía.

El estándar define varios tipos de cables y conectores según los usos que se les pretenda dar, también permite el diseño (no restringe explícitamente) de nuevos cables siempre y cuando cumplan con las restricciones de esta línea de transmisión.

USB especifica que el bus debe ser capaz de proporcionar 5 V entre las terminales Vcc y GND desde las cuales el dispositivo conectado puede tomar corriente, que al inicio no debe ser mayor a 100 mA; tal corriente se puede aumentar, si el dispositivo lo requiere, hasta un máximo de 500 mA.

Hay que mencionar que el estándar permite que se pueda implementar sistemas de manejo eficiente de energía que permiten poner el dispositivo o el bus en modo suspendido durante periodos de inactividad (e incluso apagar el dispositivo del todo).

4.4 Ancho de banda.

La especificación actual define tres tipos de velocidades para el bus serie universal:

- "Low speed": Tiene una tasa de transferencia de hasta 1.5 Mbits/s. Esta velocidad es utilizada en aquellos dispositivos que no necesitan enviar grandes cantidades de información tales como teclados, ratones, controles para juegos, etc.
- "Full speed": Con una tasa de transferencia de hasta 12 Mbits/s, antes de la aparición del USB 2.0 esta fue la máxima velocidad que se podía alcanzar. Se utiliza en dispositivos cuyas necesidades de ancho de banda sean mayores, tales como módems de mediana velocidad, micrófonos, altavoces, y algunas impresoras.
- "High speed": Actualmente la mayor de las velocidades que se puede alcanzar. Permite transmitir hasta 480 Mbits/s, lo cual la hace ideal para dispositivos de almacenamiento masivo portátiles, cámaras de alta definición, y otros dispositivos que requieran tales anchos de banda.

Se debe aclarar que estas velocidades son las máximas velocidades teóricas que se podrían alcanzar solo bajo condiciones ideales. En la práctica la velocidad real alcanzada suele ser bastante menor a lo indicado, p.e., a la fecha de realización de este trabajo la máxima velocidad que se pudo alcanzar en el modo de alta velocidad es de cerca de 30 Mbytes/s, sin embargo, la mayoría de los dispositivos comerciales catalogados como "hi-speed" trabajan a una velocidad de entre 3 y 20 Mbytes/s, que es bastante menor a la máxima velocidad que el estándar permite (60 Mbytes/s).

4.5 Tipos de transferencia y "endpoints".

"Endpoint" de control: Las transmisiones de control son utilizadas principalmente para enviar datos de control del bus, así como para comunicarse con el dispositivo USB y

obtener las tablas de descriptores. Según el estándar USB, el endpoint cero siempre es un endpoint de control y se utiliza para inicializar, enumerar y configurar el nodo. También pueden programarse otros "endpoints" para que funcionen de esta manera y sean utilizados por la aplicación cliente. En este tipo de “endpoints” tienen una detección de errores bastante robusta y cuentan con mecanismos de recuperación y retrasmisión de paquetes.

"Endpoint" sincrónico: Las transmisiones "isochronous" están diseñadas para aquellas aplicaciones donde se requiere transmisión en tiempo real como voz o vídeo. Este tipo de transmisiones no tiene control de errores, si llega un dato erróneo, este simplemente es marcado como tal, pero no se solicita una retransmisión, pues se supone que la cantidad de errores en el canal USB es suficientemente baja para que este efecto no sea considerable. La latencia en la transmisión es garantizada al igual que la velocidad. La latencia dependerá del tamaño del buffer de transmisión y recepción, y la velocidad será un parámetro por especificar en el "endpoint". Ejemplos de dispositivos que pueden utilizar este tipo de transmisión son los dispositivos de audio y vídeo.

"Endpoint" "Bulk": Las transmisiones "Bulk" tienen como objetivo alcanzar la máxima tasa de transferencia disponible en la red USB en un momento específico manteniendo la integridad de los datos, no se garantiza latencia ni ancho de banda, ya que dependerá de otras actividades en el bus. Es muy útil para transferencias de grandes cantidades de datos como sucede con los "scanners", impresoras y dispositivos de almacenamiento masivo.

"Endpoint" tipo interrupción: Finalmente las transmisiones por interrupción ("interrupt") garantizan una tasa de transmisión fija y una latencia no menor a la fijada, así

se garantiza integridad en los datos. Ejemplos de aplicaciones son los "mouses", teclados y todo tipo le dispositivos puntero o de interacción humana (HID).

Todo dispositivo USB puede tener más de un "endpoint" hasta un máximo de 16, el estándar define de manera unánime que el "endpoint" 0 debe ser un "endpoint" de control bidireccional dedicado a manejar las transferencias de control.

A través de los "endpoints" se puede transmitir grandes cantidades de datos. Si la cantidad de datos es mayor a la máxima capacidad de datos por paquete en ese específico "endpoint", entonces se procede a realizar múltiples transmisiones. Esto se hace de manera automática en el "software" USB de la PC; en el nodo dependerá de qué tan integrado sea el dispositivo interfaz USB utilizado. Un "endpoint" puede informar al "Host" que se encuentra ocupado y no puede realizar la actual petición devolviendo NAK corno respuesta, la PC simplemente tratará de realizar la comunicación más tarde.

Debido a las características del proyecto en desarrollo se decidió utilizar "endpoints" tipo sincrónico, ya que aseguran un ancho de banda constante en todo momento.

4.6 El proceso de enumeración e inicialización.

Cuando un dispositivo USB es conectado a una red USB se inicia un proceso que tiene como fin integrar el nuevo dispositivo al bus.

Este proceso comienza con la conexión del nodo a un "hub", en este momento el "hub" informa al "host" de la conexión del nuevo dispositivo y el "host" ordena la habilitación del puerto donde está conectado el nodo, luego el nodo se resetea al detectar

voltaje en el bus USB, además el nodo responde por defecto a la dirección 0; lo primero que el "host" trata de hacer es lograr asignar una nueva dirección al nodo. Para esto el "host" emite una transferencia de control "setup" "set-address" con la dirección por asignar en la transacción de datos de la transferencia, el nodo responde "ok" en el "status" en el caso de una transferencia satisfactoria. En este momento el nodo comienza a responder en la nueva dirección y se encuentra en el estado "address", todos los paquetes en la red USB que no vayan dirigidos a esta dirección serán ignorados por el nodo.

El "host" trata de conseguir información del nodo, esto lo logra mandando comandos "Setup-Get Descriptors"; así el "Host" logra obtener los descriptores de "Device", "Configuration", "String", "Interface" y "Endpoint", los cuales le sirven al "host" para luego ajustar o fijar la configuración del nodo con un "Setup-Set Configuration", "Setup-Set Feature" y "Setup-Set Interface" como mínimo para que el nodo quede funcional en el estado "configured" para que la aplicación de usuario tome el control del nodo.

Todos los pasos seguidos por el "Host" hasta este momento son realizados por el "software" y "hardware" USB, y es parte del sistema operativo.

Durante la enumeración el controlador del dispositivo negocia el ancho de banda que va a necesitar para comunicarse con este en el caso de las comunicaciones que tienen temporización. Si el ancho de banda requerido no está disponible, el "host" no permite que se inicien las comunicaciones entre el dispositivo y su controlador, entonces el controlador debe solicitar un ancho de banda menor o esperar a que esté disponible la cantidad

solicitada. En el caso de comunicaciones sin temporización, deben esperar a que el bus se encuentre disponible, y utilizar el ancho de banda disponible en ese momento.

Según el estándar, todos los dispositivos de alta velocidad deben responder ya sea a la enumeración como dispositivos de alta velocidad, 480 Mb/s, o de "full speed" 12 Mb/s, esto para asegurar la compatibilidad con el estándar anterior, sin embargo esto no significa que deba funcionar "full speed" (el FX2 puede funcionar en ambos modos).

Cuando un dispositivo tipo "high speed" se conecta, este es tratado al inicio como un dispositivo tipo "full speed", luego, durante el reseteo, el dispositivo debe negociar con el "host" el funcionamiento en modo "high speed"; si no puede establecerse como dispositivo de alta velocidad, entonces continuará respondiendo como un dispositivo tipo "full speed".

CAPÍTULO 5: El CY7C68013.

5.1 Descripción del CY7C68013.

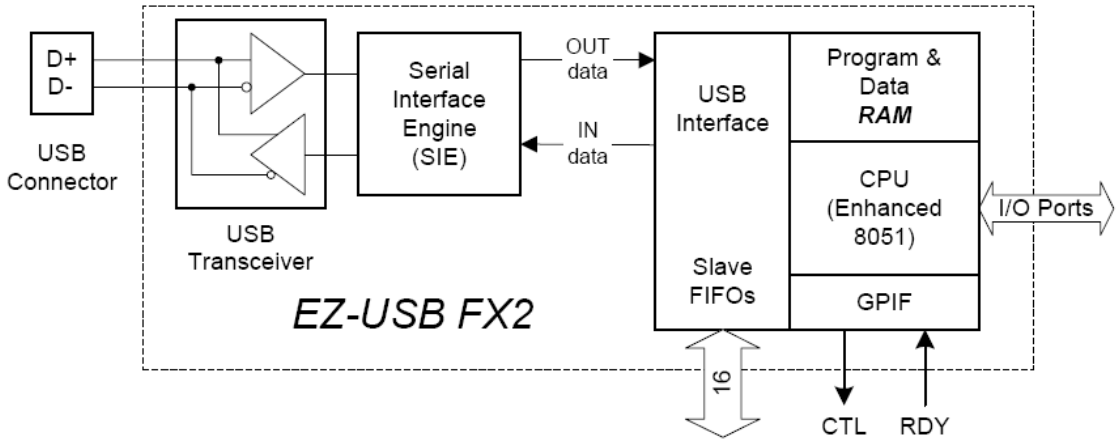


Figura 5.1.1 Diagrama simplificado del CY7C68013A en su versión de 56 pines.

El chip CY7C68013 pertenece a la familia de circuitos integrados EZ-USB FX2 de Cypress Semiconductors, esta familia permite implementar el protocolo de transmisión de datos USB en su versión 2.0, de una manera sencilla y eficiente. Básicamente el circuito integrado consiste en una máquina de interfaz serial, SIE (por sus siglas en inglés), que se encarga de implementar el protocolo USB, y un microcontrolador tipo 8051 mejorado. En

conjunto pueden implementar hasta cuatro "endpoints" que pueden operar a 12 Mbit/s o a 480 Mbits/s según se esté trabajando con un "host" en versión 1.1 o 2.0, respectivamente.

El SIE se encarga de la codificación, decodificación, corrección de errores de los datos que se encuentran almacenados en los FIFOs internos o de los datos recibidos desde el "host", así como de encargarse de asignar los niveles de voltaje adecuados para su transmisión, simplificando la tarea de transmitir datos a través del puerto.

Por otro lado el CPU es una versión mejorada del muy conocido 8051. Este micro funciona como un periférico del SIE, el controlador ejecuta el código almacenado en su RAM interna, que a su vez se carga ya sea de una memoria tipo EEPROM externa o por medio del bus USB. Como periférico cuenta con estas funciones:

- Implementar la parte del protocolo de alto nivel al atender las solicitudes del "host" sobre el "endpoint zero", "endpoint" de control, que no requieran periodos de respuesta muy rápidos.
- Servir como microcontrolador de uso general.

Se debe hacer notar que el 8051 interno puede funcionar en tres frecuencias 12, 24 y 48 MHz, mientras que el SIE funciona a 480 MHz, por lo que este micro no es capaz de encargarse de la transmisión y recepción de datos, sino que, generalmente, se limita a configurar las interfaces y luego deja el control al SIE.

Además, el sistema cuenta con varias memorias tipo “primero que entra, primero que sale”, FIFO (por sus siglas en inglés), que sirven como medio de almacenamiento temporal a los datos que entran y salen del chip. Estos FIFOs pueden ser controlados ya sea por un reloj maestro externo para operar de modo sincrónico, o por medio de una señal

de "strobe" para operar asincrónicamente. En cualquiera de los dos casos, estos solo pueden trabajar en modo esclavo, es decir, necesitan de una lógica, interna o externa, que controle el proceso de carga y envío de datos, desde y hacia el bus USB.

El chip FX2 posee una lógica interna que permite controlar estos FIFOs, la interfaz general programable, GPIF (por sus siglas en inglés), que es una máquina de estados finita programada por el usuario, permite implementar diversos protocolos de intercambio de datos al configurar las entradas y salidas disponibles para este fin; tal máquina puede funcionar ya sea con un reloj externo o con el reloj interno.

Por último se debe mencionar que el chip cuenta con varios puertos de entrada y salida que permiten el envío y recepción de datos, así como funcionar a manera de puertos de entrada/salida de uso general.

Este chip viene en presentaciones de 128, 100 y 56 pines, cada presentación varía en la cantidad de puertos de entrada/salida disponibles, así como la presencia de varias señales de control útiles, tales como los temporizadores. La figura 5.1.2 muestra las diversas configuraciones de pines disponibles.

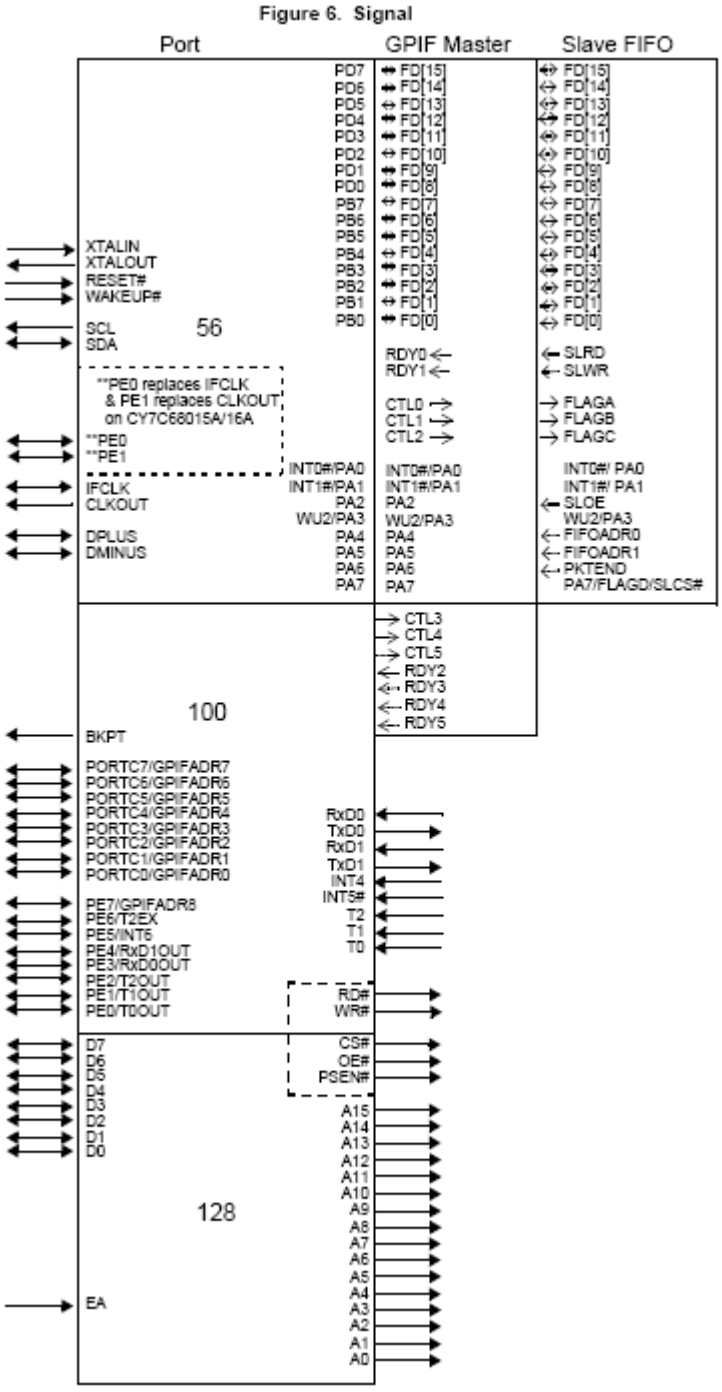


Figura 5.1.2 Diagrama simplificado del CY7C68013A en su versión de 56 pines.

5.2 El controlador 8051 embebido.

Como se ha venido mencionando, el CY7C68013 cuenta con un microcontrolador interno que posee un set de instrucciones compatible con el de un 8051 estándar en lo que respecta a la codificación binaria, sin embargo, sí se diferencia respecto al tiempo de ejecución, ya que en este caso, el FX2 tiene en promedio un incremento en el rendimiento de 2,5 veces un controlador normal de este tipo, además puede correr a una velocidad de hasta 48 MHz, cerca de cuatro veces la velocidad de los 8051 normales. Esta compatibilidad permite utilizar el código y los compiladores normales para un controlador de este tipo, por lo que no se detallará mucho acerca de esto durante el trabajo.

Aparte de tener un ciclo de instrucción menor y poseer una velocidad de reloj mayor, el FX2 incluye entre otras, las siguientes mejoras en la arquitectura:

- Un segundo puntero de datos.
- Una segunda USART.
- Un tercer contador de 16 bits.
- Una interfaz de memoria externa de alta velocidad de 16 bits no multiplexada.
- Dos autopunteros (punteros de datos autoincrementales).
- Interrupciones USB y FIFO/GPIF vectorizadas.
- "Sleep mode" con tres señales para despertar.
- Un bus I2C compatible de 100 o 400 kHz.
- Registros de funciones especiales, SFR, específicos para el FX2.

Además cuenta con una memoria RAM de 8 KB, que puede se cargada ya sea por medio del bus USB o por medio de una EEPROM externa.

5.3 El SIE.

Todas las interfaces USB cuentan con una "máquina de interfaz serial" o SIE por sus siglas en inglés, esta máquina se conecta a las líneas de datos del USB y entrega información desde y hacia el bus. Esta máquina se encarga de decodificar los paquetes, ejecutar el cheque de errores y entregar la información decodificada al dispositivo.

Una de las características que posee el FX2 es la habilidad de autoreenumerarse, permitiéndole tomar cualquier identidad.

Como ya se explicó, cuando un dispositivo USB se conecta al bus, este le envía al "host" información acerca de su fabricante, modelo, y número de serie y el "host" le devuelve un número de identificación a la vez que carga el controlador adecuado. En este caso, una vez que el FX2 se conecta se configura como un "dispositivo USB por defecto", es decir, carga las tablas que tiene almacenadas de la fábrica, con esta información el "host" busca el controlador adecuado, el cual descarga automáticamente el "firmware" y tablas de descriptores USB a la memoria RAM que el chip posee para este fin, acto seguido, el FX2 simula una desconexión/conexión haciendo creer al "host" que se ha conectado otro dispositivo, volviendo a iniciar el proceso de enumeración, pero esta vez lo hace con la información descargada, Tal proceso es denominado "ReNumerationTM", y se realiza sin que el usuario lo perciba.

5.4 Tarjeta de desarrollo PmodUSB2.

Debido a la cantidad de pines disponibles, el chip CY7C68013 solo se encuentra disponible en empaquetados de montura superficial, lo cual hace que su manejo sea bastante incómodo, por este motivo se decidió utilizar el módulo PmodUSB2 de Digilent Inc. Este módulo está diseñado para ser utilizado como un módulo periférico para otras tarjetas de esta empresa, sin embargo puede ser utilizado para cualquier otro fin.

Esta tarjeta contiene un chip CY7C68013A en su versión de 56 pines, con todos los componentes externos necesarios para su correcto funcionamiento, así como 6 diodos LED que pueden ser utilizados como indicadores; dos botones y un "jumper" que pueden ser programados como entradas del FX2. Con esto solo quedan disponibles dos puertos de ocho bits cada uno y las señales de control compartidas de estos.

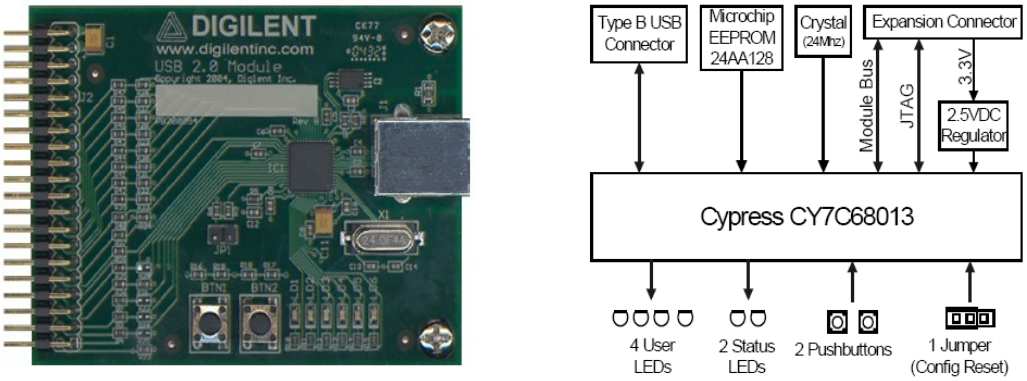


Figura 5.4.1 Fotografía y diagrama de conexión del módulo USB 2.0 de Digilent.

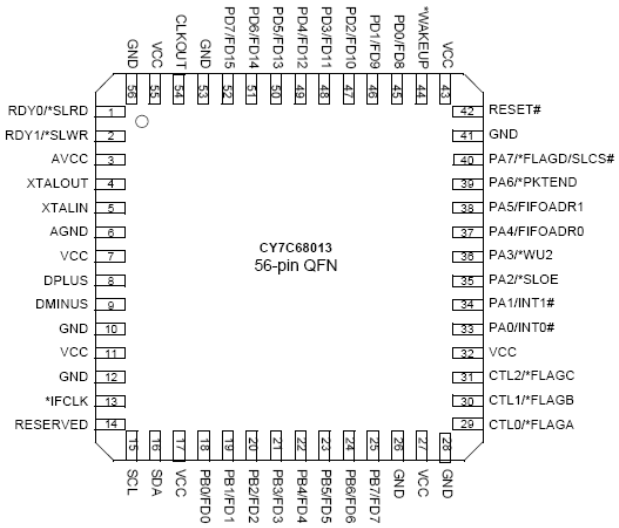


Figura 5.4.2 “Pin-out” del CY7C68013A en su versión de 56 pines.

A pesar de que el FX2 puede manejar un bus de 16 bits, debido a la presencia de los led y botones en la tarjeta, solo se dispone de un bus de datos ocho bits (PBx), y de ocho señales de control para manejar los circuitos conectados al exterior (PAx), que junto con los pines de control suman 19 pines de entrada/salida disponibles para la realización del proyecto, sin embargo, a la hora de diseñar la interfaz se debe tener en cuenta que no todas las patillas del puerto A pueden ser utilizadas como entradas o salidas de control, sino que, dependiendo del tipo de GPIF o FIFO implementado se utilizarán hasta cinco pines, dejando únicamente tres pines para el control del circuito de medición.

Además se debe señalar que la versión de 56 pines no cuenta con las señales necesarias para el manejo de los contadores/temporizadores en modo contador, por lo que estos no fueron utilizados para generar las bases de tiempo necesarias para el circuito contador, en su lugar hay segmentos de código en el firmware especialmente diseñados para este fin.

CAPÍTULO 6: Implementación y análisis de resultados.

6.1 Firmware.

La etapa de desarrollo del firmware fue la etapa que tuvo más dificultades a lo largo de todo el proyecto. Dentro de los problemas más importantes que se tuvieron que afrontar, está el hecho de tener que lidiar con las pocas señales de control que posee la versión de 56 pines del FX2LP, y se dificultaba aún más porque parte de estas estaban siendo utilizadas en alguna parte de la tarjeta, como en los botones o los leds, o no contaban con ningún tipo de conexión que facilitara su uso. También jugó un papel importante el que la mayoría de los ejemplos que el fabricante pone a disposición del público, necesitan hardware especifico, como módulos de desarrollo, así como herramientas propietarias de difícil acceso.

6.1.1 El CY7C68013A.

Con el cambio de velocidad de USB 1.1 a la versión 2.0 también se introdujo una gran cantidad de detalles que se deben tomar en cuenta a la hora de programar un dispositivo USB, por esta razón el IC FX2LP dispone de una gran cantidad de registros e instrucciones que ayudan a simplificar en gran medida la implementación de los dispositivos, además el fabricante pone a disposición de los usuarios una serie de librerías que evitan tener que programar el chip desde cero, sin embargo esto no lo convierte en una tarea trivial.

El FX2LP tiene tres modos de operación respecto al uso de los puertos:

- Modo Puertos: en este modo de operación todos los pines son utilizados como puertos de entrada salida del microprocesador y los pines con funciones especiales están deshabilitados.
- Modo GPIF (General Programmable Interface): En este modo el GPIF es el maestro de los FIFOs de los endpoints y de los periféricos, y se utilizan parte de los pines dedicados a puertos como señales de control y buses de datos para los FIFOs. GPIF se encarga de adquirir los datos de los perifericos y de mandar los paquetes al “Host”.
- Modo FIFO esclavo: En este modo los FIFOs de los endpoints son manejados por un controlador externo al chip, el cual se debe encargar de suministrar los datos e indicar al FX2LP cuando debe mandar los paquetes USB.

A parte de estos modos el FX2LP cuenta con dos modalidades para enviar los paquetes USB:

- Modo Manual: En este modo el 8051 es el encargado de recolectar la información, empaquetarla y enviarla al “Host”, esto permite tener gran flexibilidad a la hora de procesar los datos que se desean enviar, sin embargo, debido a la baja velocidad a la que opera el núcleo del microprocesador, no se pueden alcanzar grandes velocidades de transferencia.
- Modo Automático: En este caso un maestro externo al GPIF es el encargado de enviar los datos al “Host”, en este modo se pueden alcanzar grandes tasas de transferencia

ya que el 8051 no interviene en el manejo de los datos y solo se encarga de inicializar el dispositivo y los respectivos endpoints.

En el presente proyecto, a pesar de no necesitar un ancho de banda grande, se pretendía utilizar el GPIF del FX2LP para adquirir los datos del CAD, sin embargo, el uso de dicho instrumento no es una tarea trivial y se necesitan herramientas especializadas para su correcto depurado. Durante el proceso de desarrollo se presentaron inconvenientes a la hora de poner en funcionamiento dicha herramienta, ya que la mayoría de los ejemplos que se disponían estaban diseñados para ser utilizados en la versión de 128 pines y utilizando el módulo de desarrollo vendido por el fabricante, lo cual dificulta el proceso de aprendizaje ya que se imposibilitó saber si la herramienta fallaba por una implementación en el firmware o por una mala conexión en los pines. Además, la falta de fuentes de información precisa añadió dificultad al trabajo. Pese a que el fabricante pone a disposición un manual técnico de referencia, solo cubre el uso general de las diferentes utilidades del chip y en algunas ocasiones la información no es suficiente; por otra parte, la información disponible en libros de terceros y foros de desarrollo de hardware es prácticamente nula, en especial para la versión de 56 pines que se utilizó en este proyecto.

Por estas razones se decidió utilizar el FX2LP en modo puertos y de forma manual, es decir, el 8051 es el encargado de recolectar los datos necesarios y empaquetarlos de manera adecuada para su envío al “Host”.

El FX2LP cuenta con seis endpoints, cuatro de los cuales, EP2, EP4, EP6 y EP8, son altamente flexibles y pueden adaptarse a prácticamente cualquier aplicación. Los otros dos son mucho menos configurables: el EP0 que, por el estándar USB siempre debe estar

habilitado y funcionar como “endpoint” de control; y el EP1 que solo puede ser utilizado para mandar paquetes tipo “interrupt” o “bulk” que no superen los 64 bytes, por lo que no se puede utilizar dicho “endpoint” ya que el presente proyecto necesita utilizar “endpoints” de tipo “sincrónicas”.

Dado que el dispositivo que se diseñó solo sirve para enviar datos al “Host” y debe operar en modo sincrónico, se utilizó el EP6 como un “endpoint” de entrada tipo “sincrónico” con un paquete máximo de 32 bytes. Se decidió utilizar un tamaño de paquetes pequeño por dos razones, primero para evitar que el “Host” deba pasar mucho tiempo esperando por los datos y la segunda es que debido a que la información es dependiente del tiempo, es necesario mandarla al “Host” a la mayor brevedad a fin de que el usuario pueda tener los datos en tiempo real.

6.1.2 Firmware de control del contador.

El firmware de control del contador se encarga de generar la base de tiempos para el contador, capturar los datos que están disponibles en el bus y pasarlos al Endpoint correspondiente.

Debido a que el FX2 se programó como un dispositivo USB tipo sincrónico, el “Host” revisa periódicamente el dispositivo en busca de nuevos paquetes de datos para transportar, y es responsabilidad del firmware asegurar que los datos estén disponibles sin ningún tipo de retraso, de lo contrario el “Host” generaría un error de retraso en el tiempo de de respuesta (turnaround error), el cual debe ser manejado de forma especial por el “Host”. A fin de evitar esto, se decidió que el firmware debería armar el paquete de datos copiando de manera repetida en el paquete el último dato válido obtenido. De esta forma se

asegura que siempre se están enviando paquetes al “Host”, además, como precaución se habilitó el envío de paquetes nulos, es decir, si por alguna razón no hay ningún paquete listo para ser enviado, el FX2LP automáticamente generará un paquete de longitud cero, evitando el error antes mencionado.

Como se señaló anteriormente, se decidió utilizar el 8051 como generador de base de tiempos, debido en parte a que un circuito de reloj externo implicaría el uso de varias señales de control y sincronización, de las cuales se tienen muy pocas en la versión de 56 pines del CY7C68013A. A fin de ser lo más precisos posible se utilizó un código propietario brindado por el fabricante en forma compilada, que permite obtener retardos de ejecución en múltiplos de un milisegundo, con esto se pueden obtener bases de tiempos de hasta varios segundos con una precisión relativamente alta.

Dado que el menor de los retardos que se pueden obtener con esta función es de 1 ms, la máxima frecuencia teórica que se puede medir es de

$$F_{\max} = \frac{2^{14}}{1x10^{-3}s} = 16{,}38\ MHz$$

En la práctica esta frecuencia es menor debido a los retrasos que implican los tiempos de ejecución de las instrucciones para subir y bajar los pines de control.

A fin que el usuario pueda seleccionar la base de tiempos de forma interactiva se implementaron varias funciones que permiten seleccionar el tiempo base desde la computadora, estas funciones, llamadas por el fabricante “vendor commands”, permiten entre otras cosas, configurar y obtener parámetros tanto del 8051 como de la interfaz USB desde el programa CyConsole proporcionado por el fabricante. El presente proyecto cuenta con las siguientes funciones específicas para el contador

- 0xFF: Habilita el circuito contador, por defecto inicia con una base de tiempos de 1 ms
- 0xFE: Configura la base de tiempos en 1 ms
- 0xFD: Configura la base de tiempos en 10 ms
- 0xFC: Configura la base de tiempos en 100 ms
- 0xFB: Configura la base de tiempos en 1 s
- 0xFA: Para el circuito contador

6.1.3 Firmware de control del CAD.

Al igual que en el caso del firmware de control para el contador, en el caso del CAD, el 8051 se debe encargar de generar las señales de control con la debida temporalización, leer los datos y pasarlos a la interfaz USB.

El CAD AD7821TE, está diseñado especialmente para trabajar con microprocesadores facilitando su interfaz. Este chip es capaz de trabajar de dos formas: “Read mode” y “Write-Read mode”, los cuales se diferencian en cómo se generan y reciben los estímulos del CAD para indicar el inicio y fin de una conversión. Para este proyecto se seleccionó el modo de operación “Read mode”, ya que solo necesita de un pin de entrada que le indica al CAD cuándo iniciar la conversión, y un pin de salida que se pone en bajo para indicar que los datos encontrados en el bus son válidos.

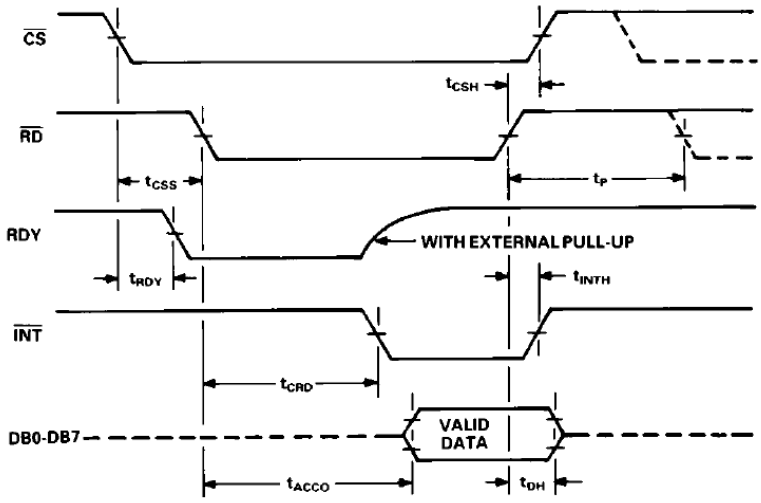


Figura 6.1.1 Diagrama de temporización del CAD en modo READ.

Tal y como se puede ver en la figura 6.1.1, la conversión se inicia al poner en bajo el pin #RD, luego, después de un tiempo tacco = 975 ns, el pin #INT se pone en bajo indicando que la salida del bus de datos es válida, estos datos estarán disponibles en el bus el tiempo que #RD permanezca en bajo. Para este caso particular el pin #CS está siempre conectado a tierra, de manera que el CAD está siempre habilitado; también se ignora la salida RDY.

Se debe hacer notar que una vez que se levanta el pin #RD, se debe esperar un tiempo mínimo de 500 ns entre cada conversión, para dar tiempo a que el circuito interno de Track-and-Hold pueda capturar un valor válido, por lo que en este modo se puede alcanzar una velocidad de muestreo máxima de 678 kMPS, o sea, se está trabajando aproximadamente al 68% de la velocidad máxima de muestreo.

Para generar el retardo entre lecturas se implementó una función utilizando instrucciones nop (No Operation), cada instrucción nop dura un ciclo de reloj, eso implica

que con un reloj de 48 MHz se deben ejecutar 24 instrucciones nop a fin de completar los 500 ns de espera entre cada lectura.

Una vez que se ha terminado la conversión, el 8051 lee los datos a través del puerto A, y los graba en la memoria correspondiente al ENDPOINT2, y una vez lleno, los envía al "Host". Para evitar esperar que el "Host" reciba los datos y libere la memoria utilizada por el "endoint", se utilizó el "enpoint" en modo "double-buffered", es decir, se cuentan dos buffers de datos para el mismo "endpoint". Así, mientras un buffer está lleno y esperando que el "Host" atienda la petición de envío de datos, el 8051 puede comenzar a procesar los datos de la siguiente transacción.

6.1.4 Firmware de depurado.

A fin de poder verificar si el CAD es capaz de proporcionar datos de manera correcta al 8051, se desarrollaron dos funciones que leen un paquete de datos de 64 bytes del puerto B y que después lo devuelven por el mismo puerto.

Estas funciones normalmente están deshabilitadas y para hacer uso de ellas es necesario introducir el "Vendor Command" 0xAA.

6.2 Tarjeta de adquisición de datos.

Las etapas de medición de voltaje, corriente y resistencia no presentaron mayores retos de construcción y puesta en funcionamiento, sin embargo, debido a la mala calidad de los componentes utilizados y a que no se utilizó ningún tipo de mecanismo para aislar las señales electromagnéticas del exterior, el ruido presente en las señales era de hasta 38 mV,

bastante alto si se toma en cuenta que la máxima resolución del CAD utilizado es de 19 mV.

El módulo de medición de capacitancias tampoco presentó mayores inconvenientes. Como era de esperarse, la linealidad del circuito es bastante buena.

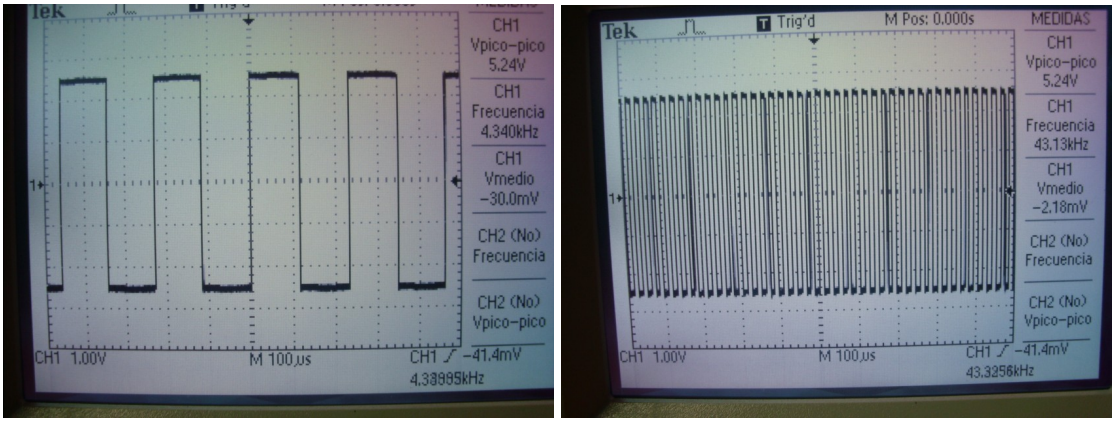


Figura 6.2.1: Salida del multivibrador para a) Cx = 0,1 µF b) Cx= 0,01 µF.

La etapa del contador presentó algunos inconvenientes debido a que la base de tiempos se implementó por medio del 8051 del FX2LP y que este debe atender otras funciones; el periodo no suele ser muy preciso. Otro problema por considerar es que los dispositivos USB que trabajan en modo sincrónico deben ser capaces de entregar al Host la cantidad de datos que se negocia durante el proceso de enumeración, pero por la naturaleza de las mediciones de frecuencia, que necesitan intervalos de medición bastante grandes comparados con los intervalos de muestreo del Host USB, hay periodos extremadamente extensos en los cuales no hay datos para enviar al Host. Esto acarrea graves consecuencias que pueden incluso detener el funcionamiento del dispositivo si no se manejan adecuadamente. Como se explicará en la sección del firmware, este problema se solucionó haciendo que el 8051 armara los Endpoints correspondientes duplicando el último valor de

la frecuencia que fue medido, y en caso de que no se haya medido ninguno se envía un paquete que contenga ceros.

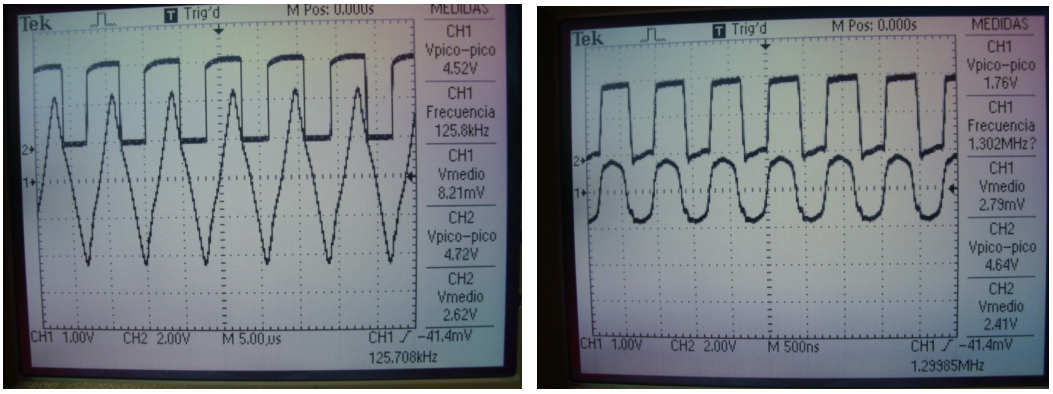


Figura 6.2.2: Salida del circuito acondicionador de señal para a) una onda triangular de 125 kHz b) una onda sinusoidal de 1.3 MHz.

6.3 Etapa de conversión analógica digital.

Como se mencionó anteriormente, para este proyecto se utilizó el CAD AD7821TE, que es una versión mejorada del AD7820, especialmente diseñado para simplificar la interfaz con microprocesadores utilizando unas pocas señales de control. Para este proyecto se probaron varias de estas configuraciones, sin embargo, debido a las pocas señales de control disponibles en la lógica de interfaz del FX2LP, se decidió utilizar el modo de lectura respecto a los otros modos disponibles ya que se puede implementar utilizando tan solo dos señales de control, una de inicio de conversión (entrada) y una señal que indica que los datos están disponibles en el bus (salida).

Una desventaja de utilizar este CI es que, debido a su bajo coste, no cuenta con ninguna clase de etapa de acondicionamiento de señal, excepto el Track-and-Hold, previa o posterior a la conversión digital, es decir que deja al usuario todo el trabajo de filtrado, tanto analógico como digital.

A fin de limitar el ancho de banda se implementó un filtro Butterworth de cuarto orden con celdas Sallen-Key, con una frecuencia de corte de 100 kHz.

La figura 6.3.2 muestra la respuesta en frecuencia en magnitud del filtro construido para esta aplicación.

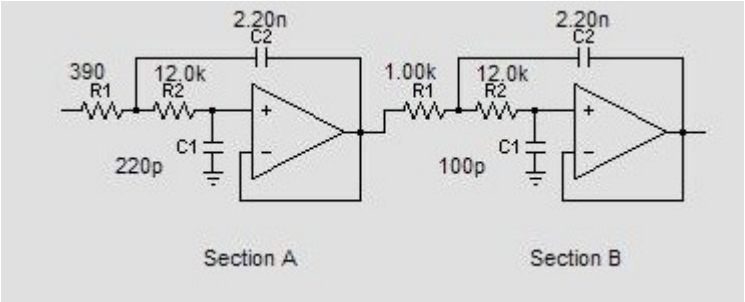


Figura 6.3.1: Esquemático del filtro pasobajo implementado.

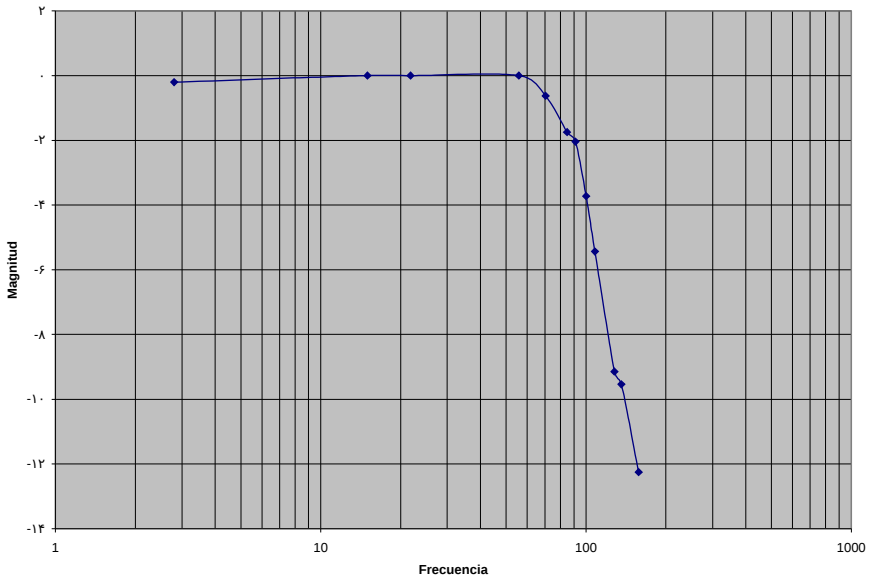


Figura 6.3.2: Gráfico de magnitud vs frecuencia para el filtro paso bajo implementado.

6.4 Ejecución y problemas encontrados.

La ejecución de este proyecto se puede dividir en tres etapas: la primera consistió en la selección de los componentes y armar el sistema de medición; la segunda etapa consintió en seleccionar los componentes necesarios para la conversión de las mediciones obtenidas por la etapa anterior en datos binarios; y la tercera consistió en seleccionar un chip para establecer el envío de datos por medio del USB y su respectiva programación.

La primera etapa del proyecto fue relativamente fácil de diseñar e implementar, ya que su complejidad es más bien baja comparada con las demás. Si bien es cierto que los circuitos no presentan mayor problema de diseño, su implementación y puesta a punto sí requirió una inversión considerable de tiempo.

La segunda etapa por su parte, sí presentó retos más serios tanto en su diseño como en su implementación. En dicha etapa se tuvo dos aproximaciones, una de las cuales resultó no ser óptima dadas la restricciones de diseño. Dado que el circuito de mediciones está en presencia de voltajes y corrientes capaces de dañar tanto el equipo de transmisión de datos (FX2LP) como el computador que recibe los datos, en una primera aproximación se intentó colocar protecciones galvánicas al sistema por medio de un IC de acoplamiento magnético. Eso requirió invertir tiempo en búsqueda y selección de un chip capaz de proporcionar un aislamiento adecuado y que a la vez fuese capaz de soportar el volumen de datos por transmitir, dicha elección no fue fácil ya que, al momento de diseñar el proyecto, la mayoría de los chips capaces de manejar los datos generados presentaban el inconveniente de tener precios prohibitivos. Además se decidió que las protecciones debían ir en la etapa digital en lugar de la analógica, ya que la mayoría de los chips de aislamiento tienen una

pésima linealidad por lo que su utilización en la etapa analógica hubiese introducido distorsiones intolerables a las señales que se deseaban medir. Por otra parte, en la etapa digital, si se selecciona un convertidor de salida paralela se tienen al menos ocho pines de salida a alta velocidad que deben contar con sendos circuitos de protección, lo cual hace que el sistema sea excesivamente caro. Por todas estas razones, en la primera aproximación se decidió utilizar un convertidor analógico de salida serial, esto resolvió el inconveniente de tener que adquirir muchos componentes de aislamiento, pero introdujo al mismo tiempo el inconveniente de tener que manejar datos seriales cuando el bus de datos de la siguiente etapa es paralelo, además con un convertidor analógico digital de tipo serial, obligatoriamente se necesita de un sistema de reloj para la etapa de conversión.

La implementación tuvo muchas complicaciones tanto en sistema de reloj como en la etapa de paralelización de datos, debido principalmente a las pocas señales de control que se disponían en el controlador principal (tercera etapa) y la dificultad de sincronizar los sistemas de reloj con la tercera etapa. Así se decidió que, dadas la restricciones de diseño impuestas por la siguiente etapa, esta aproximación no era factible y se optó por buscar otra solución al problema.

En una segunda aproximación se decidió eliminar las protecciones antes mencionadas, lo que permitió seleccionar un CAD de salida paralela simplificando enormemente el sistema de interconexión de las etapas, además por la experiencia previa con la sincronización de los relojes, en esta aproximación se seleccionó un CAD que no requiera de un reloj externo.

En la tercera etapa, se debió seleccionar un chip USB capaz de cumplir con la versión 2.0 del estándar. Así, se decidió utilizar el chip FX2LP CY7C68013A debido a que según los datos obtenidos durante la etapa de investigación previa, era el chip que brindaba mejores herramientas de trabajo y tenía un precio más accesible, sin embargo, este chip solo se encuentra disponible en empaquetados de montura superficial, lo cual dificulta su manejo por lo que se debe contar con algún tipo de tarjeta de desarrollo para simplificar el trabajo por realizar. En un principio se pretendió diseñar e implementar un circuito impreso para trabajar con este chip, pero debido a la falta de herramientas y la debida experiencia, se decidió adquirir un módulo que permitiera trabajar con dicho chip.

En el momento en que se inició el proyecto, las opciones para adquirir dicha tarjeta eran bastante reducidas, y básicamente se limitaban a la tarjeta de desarrollo del fabricante que contaba con FX2LP en su versión de 128 pines y a un módulo USB de Digilent Inc., diseñado principalmente para interconectar FPGA’s con la computadora, esta última tarjeta cuyo precio es mucho más accesible que la anterior, cuenta con el gran inconveniente de poseer la versión de 56 pines y de que, al no estar orientada al diseño de hardware nuevo, no cuenta con conectores que permitan acceder a todos los pines del chip.

De todas las etapas, la que presentó más problemas y la responsable del fallo en el proyecto fue la tercera. Una de las principales razones fue el haber seleccionado la tarjeta PmodUSB2 con el FX2LP en la versión de 56 pines, ya que esta versión cuenta con una cantidad reducida de pines de control que restringen el diseño de los sistemas de interconexión. Además, en esta versión no se cuentan con pines que faciliten la depuración, tales como los pines que permiten ver el estado de la GPIF, sin embargo, el principal

inconveniente fue el hecho de que la mayoría de los firmware de ejemplo disponibles estaban orientados a ser utilizados con el módulo de desarrollo del fabricante que cuenta con la versión de 128 pines del IC, lo cual limitó enormemente el proceso de aprendizaje y finalmente contribuyó a que no se pudiese utilizar el GPIF para adquirir los datos.

A parte de las desventajas antes mencionadas, otro factor que afectó el desempeño del proyecto fue la calidad de la fuente de alimentación utilizada para proporcionar los 3,3 V que necesita el CY7C68013A. Este circuito integrado está optimizado para disipar la menor cantidad de energía posible, por tal motivo el microcontralor interno funciona con un voltaje de 1,8 V a fin de reducir las pérdidas de energía inherentes a la conmutación de los circuitos digitales, las cuales son directamente proporcionales al cuadrado del voltaje de alimentación. Por otro lado, como es bien sabido, una reducción en el voltaje de alimentación reduce también la velocidad a la cual puede operar el circuito, por esta razón, la parte que se encarga de la transmisión de datos funciona a un voltaje de 3,3 V. Esta coexistencia de dos módulos del chip operando con distintos voltajes de alimentación hace necesario que se impongan restricciones en la rampa de voltaje de la fuente de alimentación a fin de dar tiempo a los reguladores internos de voltaje del chip para que se pongan en funcionamiento; en el caso del FX2LP este tiempo debe ser mayor a 200 µs, de lo contrario el chip puede sufrir daños irreversibles. Inicialmente, por desconocimiento de esta característica se perdieron por completo varios FX2LP en el proceso de implementación y un tercero quedó inhabilitado de hacer transacciones por el USB, ya que la fuente de alimentación se implementó utilizando un simple LM317. Esto provocó graves retrasos en

el diseño del firmware final que condujeron a que no se pudieran enviar datos a la computadora.

CAPÍTULO 7: Conclusiones y recomendaciones.

7.1 Conclusiones.

Como se observó en los capítulos anteriores, este proyecto no pudo lograr todos los objetivos propuestos. Sin embargo produjo un avance tanto en la construcción del sistema que se desea implementar, así como en el aporte de información necesaria y las consideraciones de diseño y selección de los componentes necesarios que serán de utilidad para futuros proyectos que involucren tanto el envío de datos a través del puerto USB como el diseño de sistemas a adquisión de datos.

Las principales conclusiones a las que se ha llegado después de participar en este proyecto son las siguientes:

- Se pudo comprobar que es posible construir un sistema de medición de variables eléctricas utilizando componentes pasivos de calidad media y utilizando circuitos integrados de uso común.
- Si se necesita suministrar un medio para adquirir datos a microprocesadores de velocidades intermedias, los CAD tipo half-flash son la mejor elección por su facilidad de interconexión, su precio y porque necesitan pocos elementos externos para su correcto funcionamiento.
- El puerto USB es un puerto que permite una gran flexibilidad para diseñar nuevos componentes debido a que puede ser configurado en varios modos de operación según las necesidades del hardware que se desee crear.

- El estándar USB facilita trabajo de configuración y uso de los componentes por parte del usuario al proporcionar una interfaz “Universal” para componentes de distinta índole, pero esto provoca que el proceso de diseño sea más complejo y requiere de una inversión de tiempo considerable así como la necesidad de contar con las herramientas necesarias para la depuración del componente, su software y su firmware.
- Si se desea trabajar con el CY7C68013A, debe hacerse con la versión de 128 pines, pues las otras versiones restringen las condiciones de diseño y no se prestan para la correcta depuración de los componentes de un dispositivo que pretenda cumplir con las especificaciones del estándar USB en su versión 2.0.
- Cuando se trabaja con el FX2LP se debe prestar atención a las necesidades de voltaje y corriente que tiene este chip, ya que a pesar de ser diseñado como un dispositivo de baja potencia en comparación con sus versiones anteriores, su consumo puede ser bastante alto, especialmente si se quiere trabajar con aplicaciones alimentadas por baterías; también se debe prestar atención a que la rampa de voltaje de alimentación cumpla con las limitaciones que impone el fabricante.
- Uno de los principales errores que se comenten cuando se inicia en el desarrollo de componentes USB es pretender que estos van a alcanzar la máxima tasa de transferencia, cuando en realidad la velocidad que se puede alcanzar es mucho menor, en el caso de FX2LP, la velocidad máxima teórica es de 24 Mbytes/s, cerca del 40% de la velocidad de funcionamiento del estándar, sin embargo en la práctica, los dispositivos comerciales USB 2.0 no suelen superar los 6 Mbytes/s.

- Pese a que el USB puede suministrar hasta 500 mA, se debe tener cuidado a la hora de alimentar el circuito en desarrollo por medio del bus, ya que cuando el dispositivo se conecta solo puede recibir 100 mA y si necesita más energía debe negociar con el “Host”, además no se debe olvidar que cuando el bus entra en modo “suspendido” solo puede suministrar 500 μA.

7.2 Recomendaciones.

Los principales causantes de que no se pudiesen cumplir todos los objetivos del proyecto están relacionadas con el FX2LP, si bien es cierto este chip es uno de los más fáciles de usar para implementar transacciones USB. A continuación se detallan las recomendaciones aplicables para este y futuros proyecto que involucren el puerto USB.

- Si se utiliza el FX2LP, para evitar complicaciones innecesarias a la hora de diseñar componentes desde cero, se recomienda enérgicamente utilizar la versión de 128 pines. Además de utilizar una fuente de alimentación que cumpla con los estándares dados por el fabricante.
- Durante el desarrollo del proyecto, dada la popularidad que tiene este chip, aparecieron en el mercado diferentes tarjetas de desarrollo ofrecidas por terceros, que si bien no tienen todas las características de la ofrecida por el fabricante, son mucho más flexibles que la PmodUSB2 utilizada en el proyecto y además tienen un precio bastante accesible, por esto se recomienda utilizar dichas tarjetas en proyectos futuros que involucren el puerto USB.

- Cuando se trabaja con el FX2LP, se recomienda utilizar las herramientas de desarrollo utilizadas en este proyecto en su versión completa ya que en muchos casos las versiones de prueba no son suficientes.
- Se recomienda analizar la factibilidad de desarrollar este proyecto, o una versión mejorada del mismo, con la versión 3.0 del estándar USB, la cual a la hora de finalizar este proyecto se encontraba en fase experimental, versión que permitiría alcanzar tasas de transferencia de hasta 600 Mbytes/s, por lo se podría diseñar un sistema de adquisición de alta velocidad.

BIBLIOGRAFÍA

1.Gray, N. “ABCs of ADCs, analog-to-digital basics”, **National Semiconductor tech notes**, Noviembre 24, 2003.

2.Axelson J. “USB Complete: Everything You Need to Develop Custom USB Peripherals” , **Lakeview Research**, 2005

3.Apple Inc. “USB Device Interface Guide”, **Apple Inc. tech notes,** 2007

4.Texas Instruments Inc, “Digital Logic: Pocket Data Book”, **TI Semiconductor KnowledgeBase**, 2007

5.Texas Instruments Inc, “Logic Cross-Reference”, **TI Semiconductor KnowledgeBase**, 2007

6.Cypress Semiconductor Co., “EZ-USB FX2 Technical Reference Manual v2.1”, **Cypress Semiconductor Corporation**, 2001

7.Cypress Semiconductor Co., “Cypress CyUsb.sys Programmer's Reference”, **Cypress Semiconductor Corporation**, 2003

8.Cypress Semiconductor Co., “EZ-USB® FX2™ GPIF Primer”, **Cypress Semiconductor Corporation**, 2003

9.Cypress Semiconductor Co., “Streaming Data Through Isochronous/Bulk Endpoints on EZ-USB FX2™ and EZ-USB FX2LP™”, **Cypress Semiconductor Corporation**, 2005

10.Cypress Semiconductor Co., “Introduction to the EZ-USB FX2™ GPIF Engine”, **Cypress Semiconductor Corporation**, 2003

11.Cypress Semiconductor Co., “EZ Loader Custom USB Firmware Loader Driver: AN041”, **Cypress Semiconductor Corporation**, 2004

12.Peng R., “Monitoring the EZ-USB FX2LP™ VBUS: AN15813”, **Cypress Semiconductor Corporation**, January 21, 2008

13.Gandhi S, “Implementing an 8-Bit Asynchronous Interface with FX2LP: AN6077”, **Cypress Semiconductor Corporation**, October 25, 2006.

14.Cypress Semiconductor Co., “Streaming Data Through Isochronous/Bulk Endpoints on EZ-USB FX2™ and EZ-USB FX2LP™: AN4053”, **Cypress Semiconductor Corporation**, October 24, 2005

15.Cypress Semiconductor Co., “Introduction to USB Development in Embedded Systems”, **Cypress Semiconductor Corporation**, 2008

16.Fairchild Semiconductor, “Interfacing to MM74HC High-Speed CMOS Logic”, **Fairchild Semiconductor application notes**, 1998

17.Intel Corporation, “Enhanced Host Controller Interface Specification for Universal Serial Bus Rev0.96”, **Intel Corporation**, 2001

18.Intel Corporation, “High Speed USB Platform Design Guidelines Rev1.0”, **Intel Corporation**, 2000

19.Hi-Tronic, "Keil µVision IDE ASM compile tutorial", **Hi-Tronic Inc**, 2005

20.Hi-Tronic, "Keil µVision IDE C compile tutorial", **Hi-Tronic Inc**, 2005

21.Cadsoft Computer Inc, “EAGLE manual”, **Cadsoft Computer Inc**, 2004

22.Hojas de datos varias.

23.Peacock C. “**USB in a nutshell**” http://www.beyondlogic.org/usbnutshell/usb1.htm

24.“**Universal Serial Bus**”, http://en.wikipedia.org/wiki/USB.

25.“**A Technical Introduction to USB 2.0**”

http://ww.usb.org/developers/whitepapers/usb_20t.pdf.

26.“**WinDriver™ USB User’s Manual Ver9.20**”,

http://www.jungo.com/st/support/support_windriver.html.

27.“**WinDriver™ USB Quick Start Guide**”,

http://www.jungo.com/st/support/support_windriver.html.

28.“**WinDriver USB Device™ Quick Start Guide**”,

http://www.jungo.com/st/support/support_windriver.html.

29.“**Jungo USBware USB Host Stack White Paper**”,

http://www.jungo.com/st/support/support_windriver.html

30.“**WinDriver USB Device™ White Paper**”,

http://www.jungo.com/st/support/support_windriver.html.

31.“USB Overview White Paper”

http://www.jungo.com/st/support/support_windriver.html.

32."Como hacer circuitos impresos utilizando material fotosensible".

http://www.webelectronica.com.ar/news08/nota04/nota04/placasfotosensibles.htm

APÉNDICES

Apéndice A: Firmware

```
//-----------------------------------------------------------------------
// File:      fw.c
//    Contents: Firmware frameworks task dispatcher and device request parser
//
// $Archive: /USB/Examples/FX2LP/bulkext/fw.c $
// $Date: 3/23/05 2:53p $
// $Revision: 8 $
//
//
//-----------------------------------------------------------------------
// Copyright 2003, Cypress Semiconductor Corporation
//-----------------------------------------------------------------------
#include "fx2.h"
#include "fx2regs.h"
#include "syncdly.h"            // SYNCDELAY macro

//-----------------------------------------------------------------------
// Constants
//-----------------------------------------------------------------------
#define DELAY_COUNT   0x9248*8L  // Delay for 8 sec at 24Mhz, 4 sec at 48
#define _IFREQ  48000                // IFCLK constant for Synchronization Delay
#define _CFREQ  48000               // CLKOUT constant for Synchronization Delay

//-----------------------------------------------------------------------
// Random Macros
//-----------------------------------------------------------------------
#define   min(a,b) (((a)<(b))?(a):(b))
#define   max(a,b) (((a)>(b))?(a):(b))

//Definiciones varias
#define PACKETSIZE 64
//END Definiciones varias

//-----------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------
volatile BOOL   GotSUD;
BOOL      Rwuen;
BOOL      Selfpwr;
volatile BOOL   Sleep;              // Sleep mode enable flag
WORD timebase;
BOOL COUNT_en;
BOOL CAD_en;

WORD      pDeviceDscr;    // Pointer to Device Descriptor; Descriptors may be moved
WORD   pDeviceQualDscr;
WORD   pHighSpeedConfigDscr;
WORD   pFullSpeedConfigDscr;
WORD   pConfigDscr;
WORD   pOtherConfigDscr;
WORD   pStringDscr;


//-----------------------------------------------------------------------
// Prototypes
//-----------------------------------------------------------------------
void SetupCommand(void);
void TD_Init(void);
void TD_Poll(void);
BOOL TD_Suspend(void);
BOOL TD_Resume(void);

BOOL DR_GetDescriptor(void);
BOOL DR_SetConfiguration(void);
BOOL DR_GetConfiguration(void);
BOOL DR_SetInterface(void);
BOOL DR_GetInterface(void);
BOOL DR_GetStatus(void);
BOOL DR_ClearFeature(void);
BOOL DR_SetFeature(void);
BOOL DR_VendorCmnd(void);
BOOL Get_Packet(BYTE *); //Manuel
BOOL Contador(WORD timebase, BYTE *Packet); //Manuel
// this table is used by the epcs macro
const char code  EPCS_Offset_Lookup_Table[] =
{
   0,    // EP1OUT
   1,    // EP1IN
   2,    // EP2OUT
   2,    // EP2IN
   3,    // EP4OUT
   3,    // EP4IN
```

```
  4,   // EP6OUT
  4,   // EP6IN
  5,   // EP8OUT
  5,   // EP8IN
};

// macro for generating the address of an endpoint's control and status register (EPnCS)
#define epcs(EP) (EPCS_Offset_Lookup_Table[(EP & 0x7E) | (EP > 128)] + 0xE6A1)

//-----------------------------------------------------------------------------
// Code
//-----------------------------------------------------------------------------
// Task dispatcher
void main(void)
{
  DWORD   i;
  WORD   offset;
  DWORD   DevDescrLen;
  DWORD   j=0;
  WORD   IntDescrAddr;
  WORD   ExtDescrAddr;

  BYTE Packet[PACKETSIZE];



  //
  // Initialize Global States
  Sleep = FALSE;              // Disable sleep mode
  Rwuen = FALSE;              // Disable remote wakeup
  Selfpwr = FALSE;            // Disable self powered
  GotSUD = FALSE;             // Clear "Got setup data" flag

  // Initialize user device
  TD_Init();

    // The following section of code is used to relocate the descriptor table.
    // The frameworks uses SUDPTRH and SUDPTRL to automate the SETUP requests
   // for descriptors.  These registers only work with memory locations
     // in the EZ-USB internal RAM.  Therefore, if the descriptors are located
    // in external RAM, they must be copied to in internal RAM.
    // The descriptor table is relocated by the frameworks ONLY if it is found
  // to be located in external memory.
  pDeviceDscr = (WORD)&DeviceDscr;
  pDeviceQualDscr = (WORD)&DeviceQualDscr;
                                pHighSpeedConfigDscr = (WORD)&HighSpeedConfigDscr;
  pFullSpeedConfigDscr = (WORD)&FullSpeedConfigDscr;
  pStringDscr = (WORD)&StringDscr;

   // Is the descriptor table in external RAM (> 16Kbytes)?  If yes,
  // then relocate.
   // Note that this code only checks if the descriptors START in
    // external RAM.  It will not work if the descriptor table spans
  // internal and external RAM.

    Get_Packet(Packet);
  Contador(timebase, Packet);

  if ((WORD)&DeviceDscr & 0xC000)
  {
    // first, relocate the descriptors
    IntDescrAddr = INTERNAL_DSCR_ADDR;
    ExtDescrAddr = (WORD)&DeviceDscr;
                DevDescrLen = (WORD)&UserDscr - (WORD)&DeviceDscr + 2;
    for (i = 0; i < DevDescrLen; i++)
        *((BYTE xdata *)IntDescrAddr+i) = *((BYTE xdata *)ExtDescrAddr+i);

    // update all of the descriptor pointers
    pDeviceDscr = IntDescrAddr;
                        offset = (WORD)&DeviceDscr - INTERNAL_DSCR_ADDR;
    pDeviceQualDscr -= offset;
    pConfigDscr -= offset;
    pOtherConfigDscr -= offset;
    pHighSpeedConfigDscr -= offset;
    pFullSpeedConfigDscr -= offset;
    pStringDscr -= offset;
  }

  EZUSB_IRQ_ENABLE();           // Enable USB interrupt (INT2)
  EZUSB_ENABLE_RSMIRQ();        // Wake-up interrupt

  INTSETUP |= (bmAV2EN | bmAV4EN);    // Enable INT 2 & 4 autovectoring

  USBIE |= bmSUDAV | bmSUTOK | bmSUSP | bmURES | bmHSGRANT;   // Enable selected interrupts
  EA = 1;               // Enable 8051 interrupts

#ifndef NO_RENUM
```

```
  // Renumerate if necessary.  Do this by checking the renum bit.  If it
  // is already set, there is no need to renumerate.  The renum bit will
    // already be set if this firmware was loaded from an eeprom.
  if(!(USBCS & bmRENUM))
  {
    EZUSB_Discon(TRUE);   // renumerate
  }
#endif

  // unconditionally re-connect.  If we loaded from eeprom we are
  // disconnected and need to connect.  If we just renumerated this
  // is not necessary but doesn't hurt anything
  USBCS &=~bmDISCON;

            CKCON     =     (CKCON&(~bmSTRETCH))     |
FW_STRETCH_VALUE; // Set stretch

  // clear the Sleep flag.
  Sleep = FALSE;

  // Task Dispatcher
  while(TRUE)               // Main Loop
  {
    // Poll User Device
    TD_Poll();

    // Check for pending SETUP
    if(GotSUD)
    {
      SetupCommand();         // Implement setup command
      GotSUD = FALSE;         // Clear SETUP flag
    }

    // check for and handle suspend.
     // NOTE: Idle mode stops the processor clock.  There are only two
    // ways out of idle mode, the WAKEUP pin, and detection of the USB
     // resume state on the USB bus.  The timers will stop and the
    // processor will not wake up on any other interrupts.
    if (Sleep)
    {
      if(TD_Suspend())
      {
        Sleep = FALSE;     // Clear the "go to sleep" flag.  Do it here to prevent any race condition between wakeup and the next sleep.
        do
        {
            EZUSB_Susp();          // Place processor in idle mode.
        }
        while(!Rwuen && EZUSB_EXTWAKEUP());
        // above.  Must continue to go back into suspend if the host has disabled remote wakeup
            // *and* the wakeup was caused by the external wakeup pin.

          // 8051 activity will resume here due to USB bus or Wakeup# pin activity.
          EZUSB_Resume();   // If source is the Wakeup# pin, signal the host to Resume.
        TD_Resume();
       }
     }

  }

}

BOOL HighSpeedCapable()
{
  // this function determines if the chip is high-speed capable.
   // FX2 and FX2LP are high-speed capable. FX1 is not - it does
  // not have a high-speed transceiver.

  if (GPCR2 & bmFULLSPEEDONLY)
    return FALSE;
  else
    return TRUE;
}

// Device request parser
void SetupCommand(void)
{
  void   *dscr_ptr;

  switch(SETUPDAT[1])
  {
     case SC_GET_DESCRIPTOR:                    // *** Get Descriptor
       if(DR_GetDescriptor())
         switch(SETUPDAT[3])
         {
           case GD_DEVICE:           // Device
             SUDPTRH = MSB(pDeviceDscr);
             SUDPTRL = LSB(pDeviceDscr);
             break;
```

```
            case GD_DEVICE_QUALIFIER:           // Device
Qualifier
                                                // only retuen a device
qualifier if this is a high speed
                                                // capable chip.
                                                if (HighSpeedCapable())
                                                {
                                    SUDPTRH = MSB(pDeviceQualDscr);
                                    SUDPTRL = LSB(pDeviceQualDscr);
                                                }
                                                else
                                                {

EZUSB_STALL_EP0();
                                                }
                                                break;
          case GD_CONFIGURATION:        // Configuration
            SUDPTRH = MSB(pConfigDscr);
            SUDPTRL = LSB(pConfigDscr);
            break;
           case GD_OTHER_SPEED_CONFIGURATION: //
Other Speed Configuration
            SUDPTRH = MSB(pOtherConfigDscr);
            SUDPTRL = LSB(pOtherConfigDscr);
            break;
          case GD_STRING:           // String
                                                if(dscr_ptr = (void
*)EZUSB_GetStringDscr(SETUPDAT[2]))
            {
              SUDPTRH = MSB(dscr_ptr);
              SUDPTRL = LSB(dscr_ptr);
            }
            else
              EZUSB_STALL_EP0();   // Stall End Point 0
            break;
          default:          // Invalid request
            EZUSB_STALL_EP0();      // Stall End Point 0
        }
      break;
      case SC_GET_INTERFACE:                    // *** Get
Interface
      DR_GetInterface();
      break;
      case SC_SET_INTERFACE:                    // *** Set
Interface
      DR_SetInterface();
      break;
     case SC_SET_CONFIGURATION:                 // *** Set
Configuration
      DR_SetConfiguration();
      break;
     case SC_GET_CONFIGURATION:                 // *** Get
Configuration
      DR_GetConfiguration();
      break;
    case SC_GET_STATUS:                 // *** Get Status
      if(DR_GetStatus())
        switch(SETUPDAT[0])
        {
          case GS_DEVICE:           // Device
                                    EP0BUF[0] = ((BYTE)Rwuen << 1) |
(BYTE)Selfpwr;
            EP0BUF[1] = 0;
            EP0BCH = 0;
            EP0BCL = 2;
            break;
          case GS_INTERFACE:        // Interface
            EP0BUF[0] = 0;
            EP0BUF[1] = 0;
            EP0BCH = 0;
            EP0BCL = 2;
            break;
          case GS_ENDPOINT:         // End Point
                                    EP0BUF[0] = *(BYTE xdata *)
epcs(SETUPDAT[4]) & bmEPSTALL;
            EP0BUF[1] = 0;
            EP0BCH = 0;
            EP0BCL = 2;
            break;
          default:          // Invalid Command
            EZUSB_STALL_EP0();      // Stall End Point 0
        }
       break;
     case SC_CLEAR_FEATURE:                     // *** Clear
Feature
      if(DR_ClearFeature())
        switch(SETUPDAT[0])
        {
          case FT_DEVICE:           // Device
            if(SETUPDAT[2] == 1)
                    Rwuen = FALSE;          // Disable Remote
Wakeup
            else
              EZUSB_STALL_EP0();   // Stall End Point 0
            break;
          case FT_ENDPOINT:         // End Point
            if(SETUPDAT[2] == 0)
            {
                    *(BYTE xdata *) epcs(SETUPDAT[4]) &=
~bmEPSTALL;
              EZUSB_RESET_DATA_TOGGLE( SETUPDA
T[4] );
            }
            else
              EZUSB_STALL_EP0();   // Stall End Point 0
            break;
```

```
        }
      break;
    case SC_SET_FEATURE:               // *** Set Feature
      if(DR_SetFeature())
        switch(SETUPDAT[0])
        {
          case FT_DEVICE:           // Device
            if(SETUPDAT[2] == 1)
              Rwuen = TRUE;      // Enable Remote Wakeup
            else if(SETUPDAT[2] == 2)
              // Set Feature Test Mode.  The core handles this request.  However, it is
              // necessary for the firmware to complete the handshake phase of the
              // control transfer before the chip will enter test mode.  It is also
              // necessary for FX2 to be physically disconnected (D+ and D-)
              // from the host before it will enter test mode.
              break;
            else
              EZUSB_STALL_EP0();   // Stall End Point 0
            break;
          case FT_ENDPOINT:         // End Point
                *(BYTE xdata *) epcs(SETUPDAT[4]) |= bmEPSTALL;
              break;
            default:
              EZUSB_STALL_EP0();      // Stall End Point 0
          }
        break;
      default:                  // *** Invalid Command
        if(DR_VendorCmnd())
          EZUSB_STALL_EP0();          // Stall End Point 0
    }

    // Acknowledge handshake phase of device request
    EP0CS |= bmHSNAK;
}

// Wake-up interrupt handler
void resume_isr(void) interrupt WKUP_VECT
{
    EZUSB_CLEAR_RSMIRQ();
}
```

```
//-----------------------------------------------------------------------------
//   File:      Manuel.c
//   Contents:  Hooks required to implement USB peripheral function.
//
// $Archive: /USB/Examples/FX2LP/bulkloop/bulkloop.c $
// $Date: 3/23/05 2:55p $
// $Revision: 4 $
//
//
//-----------------------------------------------------------------------------
// Copyright 2003, Cypress Semiconductor Corporation
//-----------------------------------------------------------------------------
#pragma NOIV               // Do not generate interrupt vectors

#include "fx2.h"
#include "fx2regs.h"
#include "syncdly.h"            // SYNCDELAY macro

#include "intrins.h" //Manuel

extern BOOL GotSUD;             // Received setup data flag
extern BOOL Sleep;
extern BOOL Rwuen;
extern BOOL Selfpwr;

BYTE Configuration;             // Current configuration
BYTE AlternateSetting;          // Alternate settings

#define VR_NAKALL_ON    0xD0
#define VR_NAKALL_OFF   0xD1

//Vendor commands para el CAD
#define VR_COUNT_EN                     0xFF
#define VR_TIMEBASE_1MS         0xFE
#define VR_TIMEBASE_10MS        0xFD
#define VR_TIMEBASE_100MS       0xFC
#define VR_TIMEBASE_1S                  0xFB
#define VR_COUNT_DIS            0xFA

//Vendor commands para el CAD
#define VR_CAD_EN                       0xF9
#define  VR_CAD_DIS                     0xF8
//Vendor commands para el CAD

//Vendor commands genericos
#define VR_STOP_ALL                     0xF7

//Definiciones CAD
#define StartConversion ~bmBIT0
#define PortB0_OUTEN    bmBIT0
```

```
#define PortB1_INEN                ~bmBIT1
#define EndConversion     bmBIT0
//FIN Definiciones CAD

//Definiciones Contador
#define EnFLAGS           (bmBIT2 | bmBIT3 | bmBIT4)
//Habilitar pines de control
#define STOP              ~bmBIT2                //Para el
conteo
#define START             bmBIT2          //Iniciar         el
conteo
#define CLEAN             bmBIT3          //Limpiar datos
#define NO_CLEAN          ~bmBIT3
          //Adquirir datos
#define COUNTER_H         bmBIT4
          //selecionar parte alta del contador
#define COUNTER_L         ~bmBIT4
          //seleccionar parte baja del contador
//END Definiciones Contador

//Definiciones varias
#define PACKETSIZE 64
//END Definiciones varias

//extern BYTE Packet[PACKETSIZE];
extern WORD timebase;
extern BOOL COUNT_en;
extern BOOL CAD_en;

//-----------------------------------------------------------------------
// Task Dispatcher hooks
//   The following hooks are called by the task dispatcher.
//-----------------------------------------------------------------------

void TD_Init(void)          // Called once at startup
{
  // set the CPU clock to 48MHz
  CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1) ;

  // set the slave FIFO interface to 48MHz
  IFCONFIG |= 0x40;

   // Registers which require a synchronization delay, see
section 15.14
 // FIFORESET       FIFOPINPOLAR
 // INPKTEND        OUTPKTEND
 // EPxBCH:L        REVCTL
 // GPIFTCB3        GPIFTCB2
 // GPIFTCB1        GPIFTCB0
 // EPxFIFOPFH:L    EPxAUTOINLENH:L
 // EPxFIFOCFG      EPxGPIFFLGSEL
 // PINFLAGSxx      EPxFIFOIRQ
 // EPxFIFOIE       GPIFIRQ
 // GPIFIE          GPIFADRH:L
 // UDMACRCH:L      EPxGPIFTRIG
 // GPIFTRIG

   // Note: The pre-REVE EPxGPIFTCH/L register are
affected, as well...
   //          ...these have been replaced by GPIFTC[B3:B0]
registers

 // default: all endpoints have their VALID bit set
 // default: TYPE1 = 1 and TYPE0 = 0 --> BULK
 // default: EP2 and EP4 DIR bits are 0 (OUT direction)
 // default: EP6 and EP8 DIR bits are 1 (IN direction)
 // default: EP2, EP4, EP6, and EP8 are double buffered

  // we are just using the default values, yes this is not
necessary...
 EP1OUTCFG = 0xA0;
 EP1INCFG = 0xA0;
 SYNCDELAY;                  // see TRM section 15.14
 EP2CFG = 0xA2;
 SYNCDELAY;
 EP4CFG = 0xA0;
 SYNCDELAY;
 EP6CFG = 0xE2;
 SYNCDELAY;
 EP8CFG = 0xE0;

 // out endpoints do not come up armed

  // since the defaults are double buffered we must write
dummy byte counts twice
 SYNCDELAY;
 EP2BCL = 0x80;              // arm EP2OUT by writing byte
count w/skip.
 SYNCDELAY;
 EP2BCL = 0x80;
 SYNCDELAY;
 EP4BCL = 0x80;              // arm EP4OUT by writing byte
count w/skip.
 SYNCDELAY;
 EP4BCL = 0x80;

 // enable dual autopointer feature
 AUTOPTRSETUP |= 0x01;


}


void TD_Poll(void)              // Called repeatedly while the
device is idle
{
```

```
 WORD i;
 WORD count;


 if(!(EP2468STAT & bmEP2EMPTY))
  { // check EP2 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
    if(!(EP2468STAT & bmEP6FULL))
     {  // check EP6 FULL(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is full
      APTR1H = MSB( &EP2FIFOBUF );
      APTR1L = LSB( &EP2FIFOBUF );

      AUTOPTRH2 = MSB( &EP6FIFOBUF );
      AUTOPTRL2 = LSB( &EP6FIFOBUF );

      count = (EP2BCH << 8) + EP2BCL;

      // loop EP2OUT buffer data to EP6IN
      for( i = 0x0000; i < count; i++ )
      {
          // setup to transfer EP2OUT buffer to EP6IN buffer using AUTOPOINTER(s)
        EXTAUTODAT2 = EXTAUTODAT1;
      }
      EP6BCH = EP2BCH;
      SYNCDELAY;
      EP6BCL = EP2BCL;       // arm EP6IN
      SYNCDELAY;
      EP2BCL = 0x80;         // re(arm) EP2OUT
    }
  }

 if(!(EP2468STAT & bmEP4EMPTY))
  { // check EP4 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
    if(!(EP2468STAT & bmEP8FULL))
     {  // check EP8 FULL(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is full
      APTR1H = MSB( &EP4FIFOBUF );
      APTR1L = LSB( &EP4FIFOBUF );

      AUTOPTRH2 = MSB( &EP8FIFOBUF );
      AUTOPTRL2 = LSB( &EP8FIFOBUF );

      count = (EP4BCH << 8) + EP4BCL;

      // loop EP4OUT buffer data to EP8IN
      for( i = 0x0000; i < count; i++ )
      {
          // setup to transfer EP4OUT buffer to EP8IN buffer using AUTOPOINTER(s)
        EXTAUTODAT2 = EXTAUTODAT1;
      }
      EP8BCH = EP4BCH;
      SYNCDELAY;
      EP8BCL = EP4BCL;       // arm EP8IN
      SYNCDELAY;
      EP4BCL = 0x80;         // re(arm) EP4OUT
    }
  }
}

BOOL TD_Suspend(void)          // Called before the device goes into suspend mode
{
  return(TRUE);
}

BOOL TD_Resume(void)           // Called after the device resumes
{
  return(TRUE);
}

//-----------------------------------------------------------------------
// Device Request hooks
//   The following hooks are called by the end point 0 device request parser.
//-----------------------------------------------------------------------

BOOL DR_GetDescriptor(void)
{
  return(TRUE);
}

BOOL DR_SetConfiguration(void)   // Called when a Set Configuration command is received
{
  Configuration = SETUPDAT[2];
  return(TRUE);          // Handled by user code
}

BOOL DR_GetConfiguration(void)   // Called when a Get Configuration command is received
{
  EP0BUF[0] = Configuration;
  EP0BCH = 0;
  EP0BCL = 1;
  return(TRUE);          // Handled by user code
}

BOOL DR_SetInterface(void)       // Called when a Set Interface command is received
{
  AlternateSetting = SETUPDAT[2];
```

```
  return(TRUE);          // Handled by user code
}

BOOL  DR_GetInterface(void)          // Called when a Set
Interface command is received
{
  EP0BUF[0] = AlternateSetting;
  EP0BCH = 0;
  EP0BCL = 1;
  return(TRUE);          // Handled by user code
}

BOOL DR_GetStatus(void)
{
  return(TRUE);
}

BOOL DR_ClearFeature(void)
{
  return(TRUE);
}

BOOL DR_SetFeature(void)
{
  return(TRUE);
}

BOOL DR_VendorCmnd(void)
{
 BYTE tmp;

 switch (SETUPDAT[1])
 {
   case VR_NAKALL_ON:
     tmp = FIFORESET;
     tmp |= bmNAKALL;
     SYNCDELAY;
     FIFORESET = tmp;
     break;
   case VR_NAKALL_OFF:
     tmp = FIFORESET;
     tmp &= ~bmNAKALL;
     SYNCDELAY;
     FIFORESET = tmp;
     break;
                    case VR_COUNT_EN:
                    if (CAD_en)
                    {
                            CAD_en =FALSE;
                            COUNT_en =TRUE;
                            timebase =1;
                    }
                    else
                    {
                            COUNT_en =TRUE;
                            timebase =1;
                    }
                    break;
            case VR_TIMEBASE_1MS:
                    if (COUNT_en)
                    {
                            timebase =1;
                    }
                    break;
            case VR_TIMEBASE_10MS:
                    if (COUNT_en)
                    {
                            timebase =10;
                    }
                    break;
            case VR_TIMEBASE_100MS:
                    if (COUNT_en)
                    {
                            timebase =100;
                    }
                    break;
            case VR_TIMEBASE_1S:
                    if (COUNT_en)
                    {
                            timebase =1000;
                    }
                    break;
            case VR_COUNT_DIS:
                    COUNT_en =FALSE;
                    break;
            case VR_CAD_EN:
                    if (COUNT_en)
                    {
                            COUNT_en =FALSE;
                            CAD_en = TRUE;
                    }else
                    {
                            CAD_en = TRUE;
                    }
                    break;
            case VR_CAD_DIS:
                    CAD_en = FALSE;
                    break;
               default:
        return(TRUE);
  }

  return(FALSE);
 }

 //codigo funcion
```

```
//funcion de control del contador
BOOL Contador(WORD timebase, BYTE *Packet)
{
 int i;
 BYTE Dato;
 Dato = 0x00;
 //PB2=Parar
 //PB3=Limpiar
 //PB4=SEL
 OEB |= EnFLAGS; //Habilita las banderas de control para el
contador
 OEA = 0x00;                //Habilitar puerto A como entradas

 //inicializacion
 IOB &= STOP;
 IOB |= CLEAN;
 IOB &= COUNTER_L;

 _nop_( );
 _nop_( );
 _nop_( );//espera que se propaguen las sennales de control

 //Bajar el pin limpiar

 for (i =0; i < PACKETSIZE; i +=2)
 {
  IOB &= NO_CLEAN;
  _nop_( );
  _nop_( );
  _nop_( );

  //iniciar conteo
  IOB |= START;
  EZUSB_Delay(timebase); //Espera timebase milisegundos
  IOB &= STOP;// detener el conteo

  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );

  //leer parte baja
  IOB &= COUNTER_L;
  _nop_( );
  _nop_( );
  _nop_( );
  Packet[i] = IOA;

  //Leer parte alta
  IOB |= COUNTER_H;
  _nop_( );
  _nop_( );
  _nop_( );
  Packet[i+1] = IOA;

  IOB |= CLEAN;
  _nop_( );
  _nop_( );
  _nop_( );


 }


 return TRUE;
}


BOOL Get_Packet(BYTE *Packet)
{
 int i;
//  BYTE Pack[PACKETSIZE];
//  Utilizar el puerto B para implementar el sistema de control
del CAD
//  PB0= #CS
//  PB1= #INT

 OEB |= PortB0_OUTEN;//Habilitar puerto B.0 como salida
 OEB &= PortB1_INEN; //Habilitar el puerto B.1 como
entrada
 OEA = 0x00;                //Habilitar puerto A como entradas

 IOB &= EndConversion; //Primero termina la conversión
(PIN #CS en alto). Esto se hace solo por seguridad en caso
que otra
                            //parte del código trate de
manipular este puerto

 for ( i = 0 ; i < PACKETSIZE; i++ )
 {

  //Espera 500 ns para iniciar la transacción
  ////////////////////////////////////////////////////
  //24 nop instructions at 48 MHz = 500 ns delay
  ////////////////////////////////////////////////////
  _nop_( ); /* 1 CPU cycle delay for hardware */
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );

  _nop_( );
  _nop_( );
```

```
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );

  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );

  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );
  _nop_( );

  ////////////////////////////////////////////////////
  //FIN del retardo
  ////////////////////////////////////////////////////

  //INICIO de la conversión
   IOB |= StartConversión; //Pone el pin #CS en bajo para iniciar conversión
   while ( (IOB & bmBIT1) != 0x00 ) //Espera a que el pin #INT se ponga en bajo
  {
   ;
  }

  _nop_ (); //espera un ciclo de reloj por seguridad, aunque no es necesario tampoco hace danno
  Packet[i] = IOA; //Lee la salida del CAD

  IOB &= EndConversion;
 }
 return (TRUE);
}


//-----------------------------------------------------------------------------
// USB Interrupt Handlers
//    The following functions are called by the USB interrupt jump table.
//-----------------------------------------------------------------------------

// Setup Data Available Interrupt Handler
void ISR_Sudav(void) interrupt 0
{
   GotSUD = TRUE;            // Set flag
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUDAV;         // Clear SUDAV IRQ
}

// Setup Token Interrupt Handler
void ISR_Sutok(void) interrupt 0
{
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUTOK;         // Clear SUTOK IRQ
}

void ISR_Sof(void) interrupt 0
{
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSOF;           // Clear SOF IRQ
}

void ISR_Ures(void) interrupt 0
{
    // whenever we get a USB reset, we should revert to full speed mode
   pConfigDscr = pFullSpeedConfigDscr;
       ((CONFIGDSCR xdata *) pConfigDscr)->type = CONFIG_DSCR;
   pOtherConfigDscr = pHighSpeedConfigDscr;
     ((CONFIGDSCR xdata *) pOtherConfigDscr)->type = OTHERSPEED_DSCR;

   EZUSB_IRQ_CLEAR();
   USBIRQ = bmURES;          // Clear URES IRQ
}

void ISR_Susp(void) interrupt 0
{
   Sleep = TRUE;
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUSP;
}

void ISR_Highspeed(void) interrupt 0
{
   if (EZUSB_HIGHSPEED())
   {
     pConfigDscr = pHighSpeedConfigDscr;
          ((CONFIGDSCR xdata *) pConfigDscr)->type = CONFIG_DSCR;
     pOtherConfigDscr = pFullSpeedConfigDscr;
       ((CONFIGDSCR xdata *) pOtherConfigDscr)->type = OTHERSPEED_DSCR;
   }

   EZUSB_IRQ_CLEAR();
   USBIRQ = bmHSGRANT;
}
```

```
void ISR_Ep0ack(void) interrupt 0
{
}
void ISR_Stub(void) interrupt 0
{
}
void ISR_Ep0in(void) interrupt 0
{
}
void ISR_Ep0out(void) interrupt 0
{
}
void ISR_Ep1in(void) interrupt 0
{
}
void ISR_Ep1out(void) interrupt 0
{
}
void ISR_Ep2inout(void) interrupt 0
{
}
void ISR_Ep4inout(void) interrupt 0
{
}
void ISR_Ep6inout(void) interrupt 0
{
}
void ISR_Ep8inout(void) interrupt 0
{
}
void ISR_Ibn(void) interrupt 0
{
}
void ISR_Ep0pingnak(void) interrupt 0
{
}
void ISR_Ep1pingnak(void) interrupt 0
{
}
void ISR_Ep2pingnak(void) interrupt 0
{
}
void ISR_Ep4pingnak(void) interrupt 0
{
}
void ISR_Ep6pingnak(void) interrupt 0
{
}
void ISR_Ep8pingnak(void) interrupt 0
{
}
void ISR_Errorlimit(void) interrupt 0
{
}
void ISR_Ep2piderror(void) interrupt 0
{
}
void ISR_Ep4piderror(void) interrupt 0
{
}
void ISR_Ep6piderror(void) interrupt 0
{
}
void ISR_Ep8piderror(void) interrupt 0
{
}
void ISR_Ep2pflag(void) interrupt 0
{
}
void ISR_Ep4pflag(void) interrupt 0
{
}
void ISR_Ep6pflag(void) interrupt 0
{
}
void ISR_Ep8pflag(void) interrupt 0
{
}
void ISR_Ep2eflag(void) interrupt 0
{
}
void ISR_Ep4eflag(void) interrupt 0
{
}
void ISR_Ep6eflag(void) interrupt 0
{
}
void ISR_Ep8eflag(void) interrupt 0
{
}
void ISR_Ep2fflag(void) interrupt 0
{
}
void ISR_Ep4fflag(void) interrupt 0
{
}
void ISR_Ep6fflag(void) interrupt 0
{
}
void ISR_Ep8fflag(void) interrupt 0
{
}
void ISR_GpifComplete(void) interrupt 0
{
}
```